Privacy at your Fingertips: Enabling Rapid Client-Side Operations in Fully Homomorphic Encryption

Aikata Aikata, Florian Krieger, Sujoy Sinha Roy
ISEC, Graz University of Technology, Austria
Email: {aikata, florian.krieger, sujoy.sinharoy}@tugraz.at

***Abstract*—Fully Homomorphic Encryption (FHE) allows users to offload large computations to servers without revealing the underlying data. Due to this unique feature, it is applicable to a variety of domains, including privacy-preserving Machine Learning. However, all FHE schemes have two problems- slow encryption/decryption and substantial ciphertext expansion. Thus, despite its significant potential, the practical implementation of FHE faces considerable challenges due to massive computation and communication overhead. In this work we address this gap, and propose a novel Boosted-Deflation approach to optimize client-side homomorphic encryption, leveraging bootstrapping.**

This technique minimizes ciphertext expansion and reduces the communication overhead on the server as well as the client. We also eliminate the need for encoding and decoding by the client, thereby omitting the floating-point arithmetic requirement for FHE over approximate numbers. The elegance of this technique lies in its ability to utilize the built-in FHE routines and inherently maintain security and precision guarantees. The proposed technique reduces the enc/decryption computation and communication requirements by up to 97%. We employ the proposed techniques to develop a framework for FHE client operations that is compatible with both software and hardware platforms. We conduct a comprehensive design analysis and FPGA prototyping, present ASIC synthesis results, and provide microcontroller performance evaluations. The efficient architecture design methodology demonstrates up to $76\times$ speedup compared to prior works on the same platform.

***Index Terms*—FHE, Client-side encryption, CKKS, Ciphertext expansion, Privacy-preserving computation, Hardware acceleration**

1. Introduction

Fully Homomorphic Encryption (FHE) [1] allows privacy preserving computations [2] and enables applications such as secure machine learning. Let us consider the example of smart contracts in Web3 [3] and distributed ledger technologies, which face the critical challenge of ensuring secure data management throughout transactions [4]. Conventionally, a user initiates an interaction via transfer protocols (e.g., HTTPS) to a decentralized application (DApp). This involves encryption via symmetric key schemes [5]. The data is decrypted on the server and processed by smart contracts. The resulting plain data on the server reveals the processed information. In contrast, with FHE [3], no decryption on the server is done, and all computations are performed on encrypted data, as shown in Figure 1. Thus, FHE promises privacy, aligning with the core ethos of Web3.

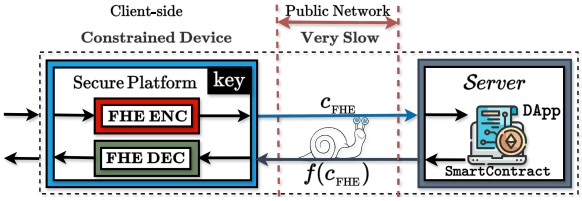

Figure 1. FHE-based Web3 Smart Contracts. The client sends encrypted data to the DApp. It remains secure during computation, and the result is returned to the client for decryption.

Despite the promise, FHE faces challenges with practical realization due to high computation and communication overhead [6], [7]. This is because homomorphic encryption converts plaintext to significantly larger ($\approx$128$\times$) polynomials, resulting in a huge ciphertext expansion problem. The practical client devices operating in these contexts are heavily resource-constrained examples include embedded systems in wearables (e.g., fitness trackers, medical sensors), low-power IoT or edge devices (e.g., smart thermostats, home security cameras). Such devices have limited computational power, memory resources, and battery life, making heavy cryptographic computations such as FHE encryption particularly challenging.

In practice, the client-side FHE software implementation for the video surveillance system [8] that sends data to the cloud for evaluation cannot encrypt or transfer even a single low-resolution video frame per second [9]. Another example is Satellite Collision Avoidance Systems using FHE [10], where constrained onboard devices must quickly homomorphically encrypt sensitive trajectory data before sending it to ground stations for secure collision-risk assessment. Minimizing encryption overhead is vital to maintaining the satellite's limited computational resources and communication bandwidth. Addressing these challenges by reducing FHE encryption overhead on constrained client devices thus becomes essential for real-world deployment of FHE's privacy-preserving capabilities across critical application domains.

1.1. Prior Works

There are two approaches for enabling FHE enc/decryption on a client device. The first approach follows the standard routines specified in FHE schemes. These routines require evaluating $2^{26}/2^{20}$ multiplications

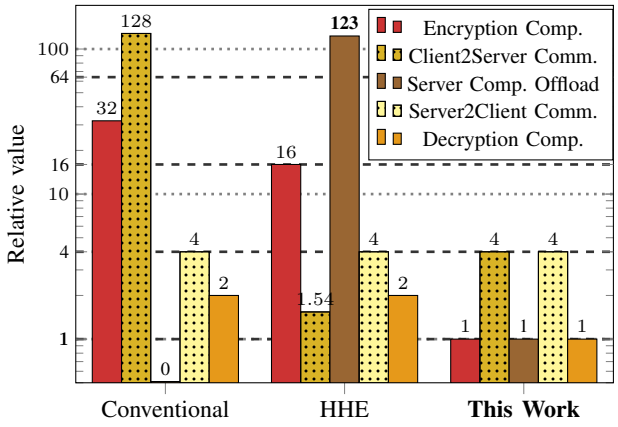

Figure 2. Illustration of how our work balances the two extremes explored in prior client-side optimization studies. The y-axis is logarithmically scaled.

(Appendix E) and floating-point arithmetic support [11], respectively. Additionally, the communication overhead for transmitting encrypted data from the client to the server is approximately 128×. A second approach has been proposed to mitigate this significant overhead, leveraging Symmetric Encryption (SE) for Hybrid Homomorphic Encryption (HHE) [7], [12], [13].

In this approach, the client encrypts plaintexts using a symmetric cipher in stream mode. The server then transforms the symmetric ciphertexts into homomorphic ciphertexts [14], [15] by homomorphically evaluating the symmetric decryption operation. Client-to-server communication decreases, but this advantage comes at the cost of increased computational overhead on the server, requiring approximately 123 bootstrapping operations [14], as illustrated in Figure 2. Notably, the client must still perform homomorphic decryption. Additionally, the server-to-client communication bandwidth remains unchanged compared to the previous approach.

As shown in Figure 2, both techniques incur significant computation overhead on the client. Therefore, a few prior works [9], [16]–[19] target high performance via compute acceleration. However, they do not consider the communication overhead. Overall, prior works either solely focused on compute acceleration or utilized SE to reduce communication overhead, introducing orders of magnitude higher computation overhead on the server.

Thus, this work aims to address the trade-off between client/server computational cost and network communication overhead. Secure video conferencing is a representative use case motivating this study [20]. While end-to-end encryption is commonly employed, audio streams are often not fully private, and the computational complexity of audio mixing grows quadratically with the number of participants. To overcome this, Beni et al. [20] apply FHE to perform audio processing in the encrypted domain, reducing the mixing complexity to linear while preserving privacy. However, their approach incurs substantial communication overhead due to ciphertext expansion, which poses a significant barrier to practical deployment, where the maximum bandwidth limit is 1MBps.

HHE has the potential to alleviate the communication overhead, but this comes at the cost of increased computational load on the server performing the audio mixing. In latency-sensitive scenarios such as real-time conferencing (≈0.25s), this overhead is a huge problem, as even minor delays can result in disruptions and loss of synchronization. Therefore, a practical solution must minimize both the communication and server-side computational overheads to be viable in real-world settings.

1.2. Our Contributions

This work presents Boosted-Deflation, a novel approach for minimizing the practical problems of computation and communication overhead for client-side homomorphic en/decryption, as shown in Figure 2. Its emphasis on simplicity, by utilizing existing FHE routines, ensures its security inherently. Our contributions are as follows.

❶ **Eliminated the need for encoding/decoding:** In a conventional setting, the plaintext is first encoded, then encrypted, and sent to the server. The resultant homomorphic ciphertext returned by the server post-computation is first decrypted and then decoded. A Discrete Fourier Transform (DFT) requiring double-precision floating-point arithmetic is utilized for encoding/decoding.

Our framework eliminates the requirement for expensive floating-point encoding and decoding on the client side. As a result, the client is relieved from performing any DFT operations before encryption or after decryption. This approach reduces the computation requirement by approximately half and saves hardware resources and energy that floating-point DFT would otherwise consume for CKKS. Our approach is particularly beneficial for microcontrollers without floating point support on the chip (e.g., Mega32 [21], Cortex-M4 [22]).

❷ **Reduced Computation Depth:** Ideal lattice-based homomorphic encryption schemes use a large ciphertext modulus, typically ranging from 438 to 1674 bits depending on parameters. Efficient implementations employ the Residue Number System (RNS) to divide this large modulus into $(L+1) > 1$ smaller prime moduli or shares. During the encryption, the client computes the ciphertext for all $(L+1)$ primes from the RNS base. For FHE, L could be 30 or larger [18], [23]. Our optimization technique leverages FHE's bootstrapping to enable encryption with just one small modulus and reduces the computation and ciphertext expansion by $\approx L + 1\times$. Additionally, the proposed technique reduces the on-chip memory/area consumption and communication overhead by $L + 1\times$.

❸ **Framework Implementation Results:** We provide a software implementation in C++ and prototype our technique on a low-cost Kintex-7 FPGA. The complete framework in software (C++) and hardware (bitfile) is open source and available at: https://github.com/aikata10/Client_Boost_Deflate. Compared to prior works, it achieves a $4.5\times$ performance increase and up to $49\times$ area reduction at $L = 30$, for FHE clients. Additionally, our design synthesized on ASIC demonstrates a $76\times$ speedup compared to prior works while maintaining the same area-to-polynomial-size ratio. We further add performance estimates using the Embedded SEAL library [24] targeting microcontrollers.

TABLE 1. CKKS PARAMETERS

Param.	Definition
N, n	Polynomial size, slots packed ($2^{16}/2^{15}$)
Δ	Scaling Factor (e.g., 2^{54})
Q, q_i	Coefficient modulus, RNS bases $Q = \prod_{i=0}^{L} q_i$
L	Max Multiplicative depth- #RNS bases - 1 (e.g., 30)
l	Dynamic Multiplicative depth ($0 \leq l \leq L$)
w	Word size ($\log q_i$=54-bit)
L_{boot}, L_{eff}	Mult. depth of/after bootstrapping (e.g., $16, 15$)

2. Background

Let $\mathbb{Z}_Q$ denote the ring of integers in the range $[0, Q-1]$. The polynomial ring $\mathcal{R}_Q = \mathbb{Z}_Q[x]/(x^N+1)$ consists of polynomials with degree up to $N-1$ and coefficients in $\mathbb{Z}_Q$. In the RNS representation [25], Q is a composite modulus made up of $L+1$ co-prime moduli q_i, defined as $Q = \prod_{i=0}^{L} q_i$. With RNS, computation modulo large Q breaks down to computations modulo q_i, allowing these smaller computations to be executed in parallel. When applying RNS, a polynomial $a \in \mathcal{R}_Q$ is transformed into a vector $\boldsymbol{a}$ of so-called 'residue polynomials', where the i-th residue polynomial is denoted as $a^i \in \mathcal{R}_{q_i}$. We use the 'monotype' font to represent ciphertexts (c) and keys (sk). The operator $\langle , \rangle$ represent dot-product operation. Noise (e) is always refreshed for every next computation to maintain security. Throughout the paper, the terms message and plaintext are used interchangeably. Polynomials in the coefficient/slot form are represented as $m(x)/m$.

2.1. FHE

Gentry [1] introduced the first FHE scheme, paving the way for significant advancements in the field over the past decade and a half. Among many FHE schemes schemes [11], [26]–[29], the CKKS scheme [11], [30] has become the most prevalent of all due to its ability to support approximate arithmetic. Consequently, most client-side works have focused on CKKS [9], [16], [18]. Therefore, our main target in this work is also CKKS. It is important to note that, in terms of computation, CKKS, BGV, and B/FV schemes share synergies, allowing the same designs to be reused.

FHE routines The FHE routines are divided into two categories: Those used by the client and the server. The server routines include homomorphic addition, subtraction, multiplication, and rotation. The client routines consist of three key operations: Key Generation, Homomorphic Encryption, and Homomorphic Decryption. In this paper, we focus on client-side FHE routines and will briefly describe them in this section. A formal description of these routines in the context of RNS CKKS [30] also follows. Table 1 describes the parameter notations. An algorithmic description of the client-server data exchange is provided in Appendix B.

Key-Generation: This is a one-time routine executed by the client at the beginning. In this routine, the client generates a pair of keys: A secret key sk and a public key pk, for decryption and encryption. Additionally, the client generates evaluation keys $\texttt{evk}_i$, used by the server during computations to ensure that the resulting ciphertext remains decryptable under the same secret key as the original ciphertexts.

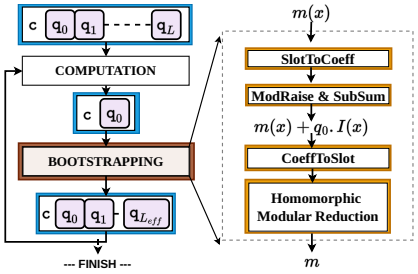

Figure 3. A high-level view of the Bootstrapping procedure used to refresh the ciphertext's multiplicative depth.

Encryption: In this procedure, first, the message m is encoded using inverse discrete Fourier transformation or IDFT. Then, it is encrypted as an RLWE (Ring Learning with Error) sample using the public key and fresh error. For simplicity, we have combined these two procedures.

• $\texttt{CKKS.Enc}(m, \texttt{pk})$: Encrypts message m, and returns $\texttt{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1) = v \cdot \texttt{pk} + (\Delta \cdot \texttt{IDFT}(m) + e_0, e_1) \in \mathcal{R}^2_{q_L}$, where $e_0, e_1/v$ are sampled from distributions- $\chi_{err/enc}$.

Decryption: The decryption routine multiplies the ciphertext with the secret key to obtain an encoded message. This is further decoded using DFT transform to obtain the resultant message. Similar to encryption, we put these two steps together in the formal description, for simplicity.

• $\texttt{CKKS.Dec}(\texttt{c}, \texttt{sk})$: Decrypts c using the secret key sk to return message $m' = \texttt{DFT}(\Delta^{-1} \cdot \langle \texttt{c}, \texttt{sk} \rangle) = f(m)$.

Bootstrapping: In CKKS FHE scheme, computations gradually reduce a ciphertext's multiplicative depth l until it reaches zero, at which point further multiplications are no longer possible. To continue, a bootstrapping (BS) procedure, shown in Figure 3, is used to refresh the ciphertext's depth. It involves four main steps. First, Slot to Coefficient Conversion transforms the ciphertext from slot form to polynomial form using a homomorphic DFT. Second, Modulus Raising lifts the modulus from q_0 to q_L, introducing a small noise term. Third, Coefficient to Slot Conversion reverts the ciphertext back to slot form using an IDFT. Finally, Homomorphic Modular Reduction removes the noise term through an approximate modular reduction. This entire BS process consumes a bootstrapping depth L_{boot}, resulting in a refreshed ciphertext with effective depth $L_{eff} = L - L_{boot}$, detailed in Appendix A.

2.2. Fast Fourier Transform and NTT

The Fast Fourier Transform (FFT) is an efficient method to compute the DFT. Defined over complex numbers, the FFT is particularly useful in reducing the complexity of DFT from $\mathcal{O}(N^2)$ to $\mathcal{O}(N \log N)$. This makes it highly relevant in FHE schemes, where efficient polynomial arithmetic is essential. NTT is FFT defined over finite fields. It is widely utilized in FHE schemes as it allows fast convolution of polynomials for polynomial multiplication, which is necessary for homomorphic encryption, decryption, and homomorphic evaluations. For more details, readers may refer to [31], [32].

3. Proposed High-level Techniques

The client-side FHE routines encounter several challenges, regardless of the implementation platform. To address these challenges, we propose two main techniques: Deflate, and Boost in the context of CKKS as a case study. The Deflate technique specifically targets the computational overhead, while the Boost technique aims to minimize both computation and communication (CoCo) overhead. Before diving into the details of the proposed techniques, we first provide an assessment of the conventional CoCo overhead in the context of RNS CKKS scheme [11], [30].

Algorithm 1 CKKS Enc.

In: $m \in \mathcal{R}, \mathtt{pk} \in \mathcal{R}^2_{q_L}$
Out: $\mathtt{c} = \text{Encrypt}(m) \in \mathcal{R}^2_{q_L}$

1: $v \xleftarrow{\$} \chi_{enc}$, $e_0, e_1 \xleftarrow{\$} \chi_{err}$
2: $\acute{m} \leftarrow \Delta \cdot \mathtt{IDFT}(m) + e_0$
3: **for** $(i = 0; i \leq L; i++)$ **do**
4: $\quad c_0^i \leftarrow (pk_0^i \cdot v_{\text{NTT}} + \acute{m}_{\text{NTT}})_{q_i}$
5: $\quad c_1^i \leftarrow (pk_1^i \cdot v_{\text{NTT}} + e_{1_{\text{NTT}}})_{q_i}$
6: **end for**
7: **return** $\mathtt{c} = \{\boldsymbol{c}_0, \boldsymbol{c}_1\} \in \mathcal{R}^2_{q_L}$

Algorithm 2 CKKS Dec.

In: $\mathtt{c}, \mathtt{sk} \in \mathcal{R}^2_{q_0}$
Out: $m' = \text{Decrypt}(\mathtt{c})$

1: $c' \leftarrow \mathtt{INTT}(\langle \mathtt{c}, \mathtt{sk} \rangle)_{q_0}$
2: $m' \leftarrow \mathtt{DFT}(\Delta^{-1} \cdot c')$
3: **return** $m' \in \mathcal{R}$

3.1. CoCo Cost Assessment

During encryption shown in Algorithm 1, the plaintext is first transformed into the polynomial form using inverse DFT and scaled by a scaling factor (Δ). Next, it is transformed to the NTT form and added to $(v \cdot \boldsymbol{pk_0} + e_0) \in \mathcal{R}_{q_L}$ to generate $\boldsymbol{c_0}$. This operation is broken into $L+1$ smaller parts in $\mathcal{R}_{q_i}$ using RNS to enhance efficiency. A similar computation is done for $\boldsymbol{c_1}$ but without the message. Note that the encoded plaintext is processed separately for each modulus q_i (base-wise), with no computation dependency across different moduli. The public key is already precomputed and split across all the $L+1$ RNS bases during the key generation procedure.

At the end of the encryption process, the ciphertext $\mathtt{c}$ contains $L+1$ pairs of residue polynomials $\in \mathcal{R}_{q_i}$ (one set each for $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$). The ciphertext is then sent to the server. After performing the necessary computations, the server brings the computational depth of the ciphertext down to zero and sends the results back to the client. Finally, one polynomial multiplication ($\langle \mathtt{c}, \mathtt{sk} \rangle$) modulo q_0 is computed during decryption, as shown in Algorithm 2.

This simplified algorithmic flow is sufficient to analyze the communication overhead. To assess the computation cost, we perform a more detailed analysis and examine the number of NTT or DFT transforms required, as this is the most expensive operation with $\mathcal{O}(N \log N)$ complexity. Since NTT/DFT and INTT/IDFT (I stands for Inverse) are equivalent in computational costs, we will refer to both as NTT/DFT for simplicity of cost comparisons.

Communication Overhead Client→Server

ⓘ The maximum number of elements that can be packed in one ciphertext is $\frac{N}{2}$.
ⓘⓘ Encryption of up to $\frac{N}{2}$ elements results in a ciphertext with $2(L+1)$ residue polynomials ($\in \mathcal{R}_{q_i}$) each of size N (total $2(L+1) \cdot N$). Hence, the communication overhead is at least $4(L+1)\times$ compared to SE ($\frac{N}{2}$).

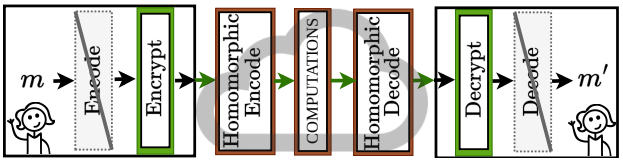

Figure 4. The simplified encryption process after applying the Deflate technique. The idea is to offload the encoding/decoding step to the server.

Communication Overhead Server→Client

ⓘ The maximum number of result elements packed in one ciphertext is $\frac{N}{2}$.
ⓘⓘ For the resultant ciphertext with 2 residue polynomials each of size N, the communication overhead is at least $4\times$ compared to symmetric key encryption.

Computation Overhead Encryption

ⓘ One DFT transform (IDFT) is required for encoding.
ⓘⓘ The encoded message, after being added to a fresh error, needs to be converted to NTT for addition with the $v \cdot \boldsymbol{pk_0}$ result. This is for one ciphertext component (c_0^i) per RNS-base. Additionally, for $\boldsymbol{c}_1$, an error polynomial is converted to NTT $\text{mod} q_i$ and added to $v \cdot \boldsymbol{pk_1}$. Thus requiring $2(L+1)$ NTT transformations.
ⓘⓘⓘ The polynomial multiplications $v \cdot \boldsymbol{pk_0}$ and $v \cdot \boldsymbol{pk_1}$ are also done using NTT for fast computation. Hence, the v polynomial components must also be converted to NTT, and $\mathtt{pk}$ is precomputed and stored in NTT format. This requires one NTT transform for all $L+1$ residues of the v polynomial. Overall, as shown in Algorithm 1, $3(L+1)$ NTT and one DFT computations are performed during encryption.

Computation Overhead Decryption

ⓘ The ciphertext component $\boldsymbol{c}_1 \in \mathcal{R}_{q_0}$ is multiplied with the secret key and added to $\boldsymbol{c}_0 \in \mathcal{R}_{q_0}$. This result is in the NTT domain; hence, an INTT over one modulus is done.
ⓘⓘ One final DFT transform is required to obtain the decoded plaintext. Thus, overall one NTT and one DFT are computed.

3.2. Deflate Technique

The key idea behind this technique is to leverage the fact that the DFT transformation matrix is public and the server performs a similar transform for bootstrapping ($\mathtt{CoeffToSlot}, \mathtt{SlotToCoeff}$). Therefore, we can offload the DFT transformation required for the encode-decode step to the server, as shown in Figure 4. This reduces the client's computational overhead and offers additional advantages, such as increased packing flexibility in implementation. By applying Deflate, the encryption and decryption routines can be rewritten as follows.

- $\mathtt{Enc}(m, \mathtt{pk})$: It encrypts m , and returns $\mathtt{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1) = v \cdot \mathtt{pk} + (\Delta \cdot \cancel{\mathtt{IDFT}}(m) + e_0, e_1) \in \mathcal{R}^2_{q_L}$.
- $\mathtt{Dec}(\mathtt{c}, \mathtt{sk})$: It decrypts $\mathtt{c}$ using the secret key $\mathtt{sk}$ to return message $m' = \cancel{\mathtt{DFT}}(\Delta^{-1} \cdot \langle \mathtt{c}, \mathtt{sk} \rangle)$.

Explanation. In CKKS, the plaintext is initially represented as a vector of elements in *slot form*. During encoding, this vector is mapped to a polynomial representation through an inverse discrete Fourier transform (IDFT).

However, during encryption the ciphertext is internally processed using the NTT representation, and the resulting ciphertext effectively stores the plaintext in slot form rather than polynomial form.

This distinction becomes important during bootstrapping. In the bootstrapping procedure, the ciphertext must first be converted from slot representation to polynomial representation using the homomorphic transform `SlotToCoeff`. After the remaining bootstrapping computations are completed, the inverse transformation `CoeffToSlot` restores the ciphertext back to slot form.

These two transformations correspond to homomorphic evaluations of the discrete Fourier transform and its inverse. In other words, `SlotToCoeff` and `CoeffToSlot` are linear transformations implemented as matrix–vector multiplications over the ciphertext slots. Importantly, CKKS encoding and decoding (IDFT/DFT) are *public linear transformations* applied to messages after a public embedding into a valid ring element. Because these transformations are linear and publicly known, they can be evaluated homomorphically without revealing the underlying plaintext.

Based on this observation, the *Deflate* technique proposes to offload the pre-encryption encoding and post-decryption decoding steps to the server. Concretely, the server performs the corresponding linear transform homomorphically using a matrix–vector multiplication, where the ciphertext acts as the vector and the matrix contains the appropriate roots of unity in plaintext form for the DFT transform. To improve efficiency, we employ the baby-step giant-step (BSGS) technique [33] (described in Appendix C), which reduces the number of ciphertext rotations and multiplications required for evaluating these linear transforms.

This design is justified by the homomorphic property of the encryption scheme. In conventional CKKS usage, the client computes

$$\mathtt{c} = \mathtt{Enc}(\mathtt{Encode}(m), pk).$$

With Deflate, the client instead sends

$$\mathtt{c} = \mathtt{Enc}(m, pk),$$

and the server subsequently applies the homomorphic encoding transform. Because encoding is a public linear operation, performing encoding before encryption is equivalent to encrypting first and then applying the encoding homomorphically:

$$\mathtt{Enc}(\mathtt{Encode}(m), pk) \approx \mathtt{FHE\_Encode}(\mathtt{Enc}(m, pk)).$$

More generally, for any server-side computation $f(\cdot)$ we obtain

$$f(\mathtt{Enc}(\mathtt{Encode}(m))) = f(\mathtt{FHE\_Encode}(\mathtt{Enc}(m))).$$

A similar argument applies to decoding. Instead of the conventional sequence

$$\mathtt{Decode}(\mathtt{Dec}(c, sk)),$$

we can equivalently apply the homomorphic decoding transform before decryption:

$$\mathtt{Decode}(\mathtt{Dec}(c, sk)) \approx \mathtt{Dec}(\mathtt{FHE\_Decode}(c), sk).$$

Therefore, both encoding and decoding can be safely shifted to the server side without exposing plaintext.

It is important to note that evaluating this transformation on the server consumes a multiplicative depth of approximately three (or four, depending on parameter choices) due to the matrix–vector multiplication required for the DFT transform. Consequently, the ciphertext initially sent for computation must retain sufficient remaining depth to support the homomorphic encoding step when applying the *Deflate* technique. However, as discussed later, this requirement can be eliminated when combined with the Boost technique.

Security Analysis. The CKKS [11] encryption is applied to $\Delta \cdot \mathrm{Encode}(m)$, while in Deflate, we directly encrypt $\Delta \cdot m$ (bypassing Encode). Note that, Encode and Decode are public linear-transformations. Therefore, if an adversary can recover m from Deflate-ciphertext, they can also recover $\mathrm{Encode}(m)$, and consequently m, in standard- Thus, breaking Deflate-ciphertext implies breaking standard- The scheme's overall concrete post-quantum security remains 128-bit (testing-script using lattice-estimator [34] is provided in the repository).

Next, we show that the encryption noise remains bounded by $B_{enc} = 8\sqrt{2} \cdot \sigma N + 6\sigma N + 16\sigma hN$ (as in standard-CKKS [11]), where σ is the standard-deviation of the Gaussian-distribution used to sample error-polynomials, N is polynomial-size, and h is the number of non-zero coefficients in the secret-key. Thus, asserting that the use of Deflate does not alter the noise bounds of the original scheme.

Proof. Consider the decryption equation of a ciphertext $\mathtt{c} \leftarrow v \cdot \mathtt{pk} + (m + e_0, e_1)$.

$$\begin{aligned}\langle \mathtt{c}, \mathtt{sk} \rangle \pmod{q_0} &= \boldsymbol{pk_0} \cdot v + m + e_0 + (\boldsymbol{pk_1} \cdot v + e_1) \cdot \mathbf{s} \\ &= m + e_{pk} \cdot v + e_0 + e_1 \cdot s\end{aligned}$$

The upper bound (B_{enc}) for the encryption noise ($e_{pk} \cdot v + e_0 + e_1 \cdot s$) can then be established as follows:

$$\begin{aligned}B_{enc} &= \|\langle \mathtt{c}, \mathtt{sk} \rangle - m \pmod{q_0}\|_\infty^{\text{can}} \\ &= \|e_{pk} \cdot v + e_0 + e_1 \cdot s\|_\infty^{\text{can}} \\ &\le \|e_{pk} \cdot v|_\infty^{\text{can}} + \|e_0\|_\infty^{\text{can}} + \|e_1 \cdot s\|_\infty^{\text{can}} \\ &\le 8\sqrt{2} \cdot \sigma N + 6\sigma\sqrt{N} + 16\sigma\sqrt{hN}\end{aligned}$$

Encryption of the encoded message additively increases the noise-bound by $\frac{N}{2}$. When we remove the encoding to support Deflate, the original noise-bound ($B' = B_{enc}$) stays intact.

Impact on CoCo.

ℹ *Decreased Encryption/Decryption Computation overhead*: The computations during decryption are reduced by half, as only one NTT transform is needed. No DFT transformation is required during decryption or encryption.
ℹ The Client ↔ Server communication overhead is unaffected.

Other Notable Advantages. Apart from the CoCo advantages, this approach offers more benefits than the conventional method. The first is variable DFT Transform size. The size of the DFT transform varies depending on the number of plaintext elements. While the maximum is $\frac{N}{2}$, the minimum is one. Therefore, a much smaller

DFT transform can be used in conventional settings for plaintexts with fewer elements for faster computation. While this is easier to implement in software, enabling this flexibility is costly in hardware due to multiple configurations. Yet, offloading this computation to the server through the Deflate technique allows for arbitrary packing flexibility while reducing computational costs.

Another advantage is eliminating the floating-point arithmetic support required for the DFT transformation. In prior work [16] supporting full client-side encryption, a dedicated floating-point arithmetic unit contributes to ≈30% area overhead on an FPGA platform. By offloading the DFT transform to the server, we eliminate the need for this expensive support on the client side. This is beneficial for edge devices such as many microcontrollers that do not support floating point arithmetic (e.g. Mega32 [21], Cortex-M4 [22], etc.). Additionally, omitting DFT reduces power consumption as floating point operations are power hungry [35]. Hence, by leveraging Deflate technique, we significantly reduce the computational burden on the client while maintaining flexibility and efficiency in the encryption and decryption processes.

Impact on Precision. The homomorphic evaluation of the encode and decode steps does not result in additional precision loss. This is because these steps are integral parts of the bootstrapping routine. Therefore, the precision provided remains consistent with the post-bootstrapping precision specified by the scheme parameters. This ensures that the overall precision of the computations is maintained and the limit is set by the bootstrapping procedure.

Computational Overhead on the Server. Deflate offloads all DFT transform computations to the server. The time consumption for these transforms depends on the parameter selection and packing. Still, it is less than the cost of one bootstrapping operation, which involves similar transforms and additional computations. ML applications often require multiple bootstrapping operations (e.g., 96 for Logistic Regression Training [36], [37] or 7,000 [23] for DNN training), making the impact of a single additional DFT minimal. Thus, the Deflate technique significantly enhances client-side operations by leveraging the server's computational power for the more complex DFT transforms.

3.3. Boost Technique

This idea is to leverage bootstrapping to reduce CoCo overhead on the client, providing a more efficient solution for secure computations. By following this technique, the decryption routine remains unchanged, and the encryption routine (Section 2.1) can be rewritten as follows.

- $\texttt{CKKS.Enc}(m, \texttt{pk})$: It encrypts m, and returns $\texttt{c} = (\boldsymbol{c}_0, \boldsymbol{c}_1) = v \cdot \texttt{pk} + (\Delta \cdot \texttt{IDFT}(m) + e_0, e_1) \in \mathcal{R}^2_{q_L \; q_0}$.

Explanation. The dynamic computational depth l of a ciphertext directly depends on the available RNS bases $(l+1)$ in the ciphertext. Due to this limitation, the initial proposal of schemes was considered levelled (SHE), as they could not support arbitrary computation depth. Gentry's seminal work [1] proposed a method to bootstrap the ciphertext, allowing for unlimited computation depth, which was subsequently applied to all levelled schemes.

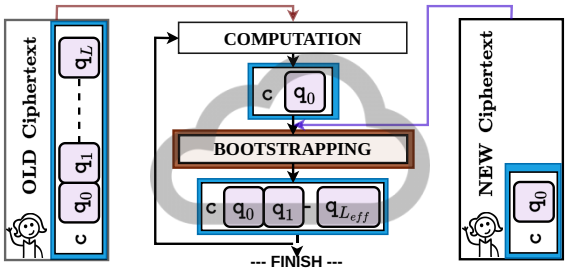

Figure 5. A high-level depiction of the key idea behind the proposed Boost technique. The 'OLD' and 'NEW' keywords show the ciphertext and data flow for the conventional and our proposed techniques.

While bootstrapping has significantly advanced server-side computing capabilities, it has not been utilized to reduce the client's CoCo overhead. In this work, we bridge this gap, and instead of sending a ciphertext with maximum computation depth, we suggest transmitting a ciphertext at zero computation depth (one RNS base only). The encryption process does not mix different RNS bases, as mentioned in Section 3.1. Thus, the CoCo requirements linearly depend on the number of RNS bases in the ciphertext. Encrypting for zero computational depth implies adding the encoded message to the product $v \cdot \texttt{pk}^2 \in \mathcal{R}_{q_0}$ (corresponding to a single RNS base).

Although this ciphertext cannot be used for computations directly, bootstrapping can be employed on the server side to increase its depth, enabling it to be processed as needed. Since the form of encrypted ciphertext did not change, the decryption routine stays as it is. This approach, as illustrated in Figure 5, significantly reduces the CoCo overhead on the client side.

Security Analysis. The Boost technique reduces the initial ciphertext modulus, however this does not negatively impact the security. This is because, the underlying RLWE security assumption [38] imposes an upper bound on the ciphertext modulus to prevent excessive noise growth, but it does not require a strict lower bound. Thus, reducing the ciphertext modulus does not weaken this assumption, since the noise levels remain within secure thresholds. The total ciphertext modulus (upper bound) required to maintain security [39] is inherently constrained once the ciphertext dimension is fixed during encryption. Furthermore, the bootstrapping process inherently maintains security under the RLWE assumption by refreshing noise.

Impact on CoCo.

ⓘ *Reduced Client → Server Communication Overhead*: The communication overhead from the client to the server is now equivalent to that from the server to the client, as the ciphertext in transit always has zero computational depth. This results in $(L+1)\times$ reduction in communication overhead.

ⓘⓘ *Decreased Encryption Computation overhead*: The computations during encryption are also reduced by a factor of $(L+1)\times$ in terms of NTT operations. This is because only the residue polynomials associated with a single RNS modulus q_0 must be processed instead of processing $(L+1)$ moduli.

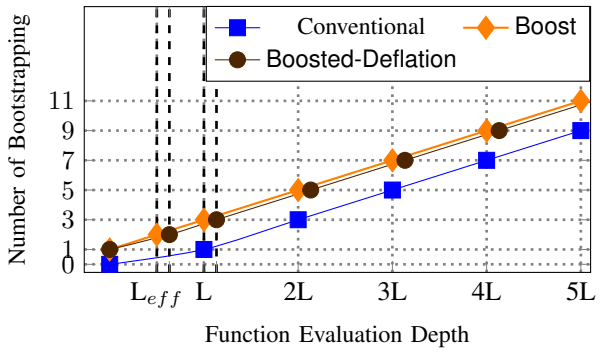

Figure 6. The number of bootstrapping operations performed on the server side increases by a constant 2 for high-depth computations ($> L_{eff}$) and 1 for lower-depth functions. For the combined Boosted-Deflation technique, the offset is lowered, allowing more computations per bootstrapping.

The Server → Client communication overhead and the decryption computation remain unaffected.

Impact on Security. To address potential security concerns, it is crucial to highlight that the strength of this proposed technique lies in its use of existing bootstrapping operations enabled by FFHE schemes. Although this represents the first instance of applying such operations on the client side, they are executed under the same security assumptions as when performed on the server side. Importantly, the bootstrapping process does not compromise privacy guarantees.

Impact on Precision. There are several works in the literature analyzing bootstrapping for various FHE schemes [40]–[42]. Our technique imposes no limitations on the type of bootstrapping used, allowing for the utilization of high-precision bootstrapping [41]. Since we do not introduce any additional or new operations, the precision of our technique is bounded solely by the bootstrapping precision of the underlying FHE scheme.

Computational Overhead on the Server. This technique reduces the computation overhead for the client, but it requires the server to perform one additional bootstrapping operation. Since the depth after bootstrapping is less than the original depth ($L_{eff} < L$) and $L_{eff} \geq \frac{L}{2}$, applications requiring high-depth computations ($> L_{eff}$) will necessitate one more bootstrapping operation. Consequently, the server's computation overhead will be at least one and at most two bootstrapping operations, as illustrated in Figure 6. Many useful applications require numerous bootstrapping operations (e.g., [23] DNN training benchmark necessitating 7000 bootstrapping), making the impact of two additional bootstrapping operations minimal.

3.4. Boosted-Deflation Technique

While our two proposed techniques offer several standalone advantages, they have conflicting requirements. The Boost technique proposes encrypting with zero computational depth, while the Deflate technique requires sufficient depth ($l = 3$ or 4 depending on the parameters [43]) post-encryption for the server to compute homomorphic encoding. However, when combined, these techniques complement each other, enhancing their respective benefits as shown in Figure 7. After applying both techniques, the final encryption/decryption procedures are as follows (shown in Table 2).

TABLE 2. A DESCRIPTION OF THE DATA AND COMPUTATION FLOW WITH Boosted-Deflation. (ALGORITHM IN APPENDIX F)

Client	Server
Encryption	
$(e_0, e_1), v$ are refreshed $\boldsymbol{c_0} = v_{\text{NTT}} \cdot \texttt{pk}_0 + (\Delta \cdot m + e_0)_{\text{NTT}} \bmod q_0$ $\boldsymbol{c_1} = v_{\text{NTT}} \cdot \texttt{pk}_1 + (e_1)_{\text{NTT}} \bmod q_0$ Send $\texttt{c} = (\boldsymbol{c_0}, \boldsymbol{c_1})$ →	→ Receive $\texttt{c}$
Server performs Function Evaluation	
	▷ Boosted-Deflation $\texttt{c}' = \text{Small-Boot}(\texttt{c})$ $\texttt{c}'' = f(\texttt{c}')$ ▷ Eval. $\acute{c} = \texttt{SlotToCoeff}(\texttt{c}'')$
Receive $\acute{c} = (\acute{\boldsymbol{c}}_0, \acute{\boldsymbol{c}}_1)$ ←	← Send $\acute{c}$
Decryption	
$m' = \Delta^{-1}((\acute{\boldsymbol{c}}_1 \cdot \texttt{sk} + \acute{\boldsymbol{c}}_0)_{\text{INTT}} \bmod q_0)$	

- $\texttt{CKKS.Enc}(m, \texttt{pk})$: It encrypts m, and returns $\texttt{c} = (\boldsymbol{c_0}, \boldsymbol{c_1}) = v \cdot \texttt{pk} + (\Delta \cdot \cancel{\texttt{IDFT}}(m) + e_0, e_1) \in \mathcal{R}^2_{q_L \; q_0}$.
- $\texttt{CKKS.Dec}(\texttt{c}, \texttt{sk})$: Returns message $m' = \cancel{\texttt{DFT}}(\Delta^{-1} \cdot \langle \texttt{c}, \texttt{sk} \rangle)$.

Explanation. Directly combining both techniques would involve initially encrypting with a computational depth of at least three, thereby encrypting with at least four moduli. Thus increasing computations by up to $4\times$ and eliminating only one DFT during encryption. Subsequently, the ciphertext will be sent to the server, where the first homomorphic encode step required by the Deflate technique will be performed. Following this step, the ciphertext will recede to a computational depth of zero, and bootstrapping will be executed to obtain additional computational depth. Finally, a homomorphic decode will be performed after the desired computations.

Overall, the CoCo overhead would be $4\times$ more when the two techniques are put together naively. This overshadows the advantages of using Deflate technique. However, upon more detailed analysis, we find that during the bootstrapping step, the $\texttt{SlotToCoeff}$ transform is initially required to convert the ciphertext from plaintext slots to polynomials. This step essentially functions as a homomorphic decode necessary to reverse the encoding performed before encryption. We can omit this step since we are conducting encoding post-encryption as per Deflate technique. In other words, the homomorphic encode performed for the Deflate technique and the first step of the bootstrapping operation ($\texttt{SlotToCoeff}$) cancel each other out and can thus be omitted. Thus, as depicted in Figure 7, a ciphertext can be sent directly for bootstrapping without necessitating an initial encoding or decoding step. The encoding will take place before the desired function evaluations start, and decoding will occur at the end before sending the ciphertext for decryption.

Impact on CoCo.

Reduced Client → Server Communication Overhead: The communication overhead from the client to the server stays reduced by $(L + 1)\times$, as the ciphertext expansion reduced from $4(L + 1)\times$ to only $4\times$.

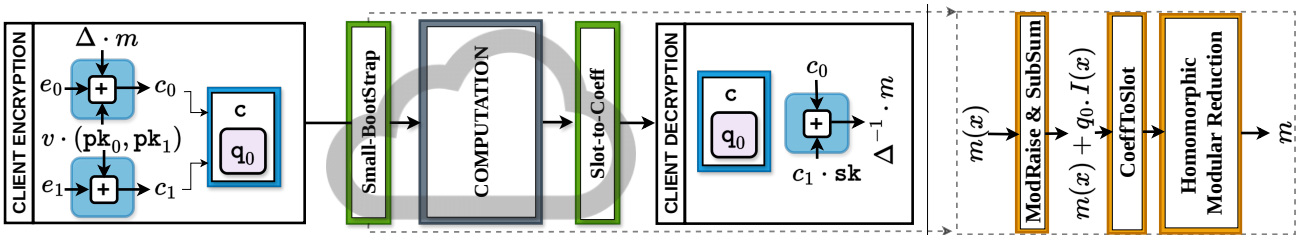

Figure 7. The operation flow following the proposed Boosted-Deflation technique. The client performs zero-depth encryption without encoding. The server performs the small bootstrapping step where the initial `SlotToCoeff` operation is omitted, and hence, it costs only 72% of the original bootstrapping. This helps save more computational depth post-bootstrapping. The server also performs the homomorphic decode (`SlotToCoeff`), so the client only has to compute decryption.

(ii) *Decreased Encryption Computation Overhead*: The encryption computation is reduced to three NTTs only.
(iii) *Decreased Decryption Computation Overhead*: The computations during decryption are reduced to one NTT instead of one NTT and one DFT.

Impact on Precision. The proposed combination of the two techniques only reduces the extra homomorphic encode/decode transformations. Hence, the proposed technique does not lead to additional loss of precision. Users must be cautious regarding the implementation parameters for overall precision.

Computational Overhead on the Server. By removing homomorphic encode and decode steps, the computation offloaded to the server is reduced. Therefore, the server only performs computations equivalent to those for the Boost technique. This effectively makes Deflate technique cost-free while significantly reducing the client's computational overhead, as depicted in Figure 6.

3.5. Security Analysis

We now provide concrete evidence to support the claim that the proposed technique poses no security threat. Intuitively, this follows from the fact that Boosted-Deflation strictly adheres to existing CKKS routines.

❶ Boosted-Deflation **technique analysis**. The combined use of Deflate and Boost results in Deflate becoming computationally free for the server, as no additional modifications are required to support it. This optimization pertains strictly to computational efficiency rather than security; thus, their combination does not alter the underlying security assumptions of the FHE scheme. For concrete definitions, readers may refer to Appendix F. Like all RLWE-based schemes, CKKS is also conjectured to be quantum-secure as it is based on the Post-Quantum secure Ring-Learning-With-Error problem. Boosted-Deflation preserves CKKS structure– ciphertexts, keys, and noise-distributions remain unchanged. Thus, our technique inherits CKKS's quantum-resistance.

Given the dynamic landscape of cryptographic attacks, parameters and secret-key distributions continue to evolve, impacting bootstrapping routines. Our proposed technique is built upon two bootstrapping subroutines- `CoeffToSlot` and `SlotToCoeff`. While the security parameters influence these underlying routines, the applicability of Boosted-Deflation remains unaffected. The proposed Boosted-Deflation explicitly demonstrates adaptability by showing that security parameters do not need an update, highlighting the key strength of our work. Thus, the generalized nature of our approach ensures robustness against evolving cryptographic standards and various secret-key distribution requirements listed in the recent Security Guidelines for Implementing Homomorphic Encryption [44].

❷ **Ciphertext Indistinguishability & IND-CPA, IND-CPA^D Security**. The CKKS encryption scheme derives its security from the hardness of the RLWE problem [38], ensuring indistinguishability under chosen-plaintext attacks (IND-CPA). It guarantees that an adversary cannot distinguish between the encryptions of two chosen messages, as the ciphertexts are computationally indistinguishable under the RLWE assumption. The Deflate technique does not alter the RLWE structure, the error term, or the statistical distribution of the ciphertexts. As a result, it does not affect the security.

A less obvious but important benefit is that enforcing bootstrapping as the first step on the server could offer protection against IND-CPA^D attacks, introduced in [45]. This attack relies on the fact that the result of CKKS decryption is not m, but instead $\Delta \cdot m + e$. Suppose this error is the same as the error used for encryption, for example, when the server evaluates an identity function. In that case, encryption security ($-a \cdot s + e + \Delta \cdot m$) does not hold, as a is public, and an attacker can recover secret s. Hence, in the IND-CPA^D threat model, CKKS is not secure. However, the authors list bootstrapping as a possible precautionary countermeasure. Note than any countermeasure technique for the attacks in the IND-CPA^D thread model (e.g., [46]) remains applicable with Boosted-Deflation, due to its use of existing homomorphic routines.

3.6. Ablation study

Performance Comparison of the various techniques are provided in Table 3. The results for the client are provided using our Software framework (discussed in Section 4.2 ❶). It takes 9.7s to compute one CKKS-bootstrapping (`Coeff2Slot` - 2.15s, `SlotToCoeff` - 1.07s). The server-side results are provided using FPGA-accelerator [47], which can perform the bootstrapping in 15.3ms, with `CoeffToSlot`, and `SlotToCoeff` comprising approximately 3.62ms, and 1.78ms, and consuming a computational depth of four each. The table shows how the combined Boosted-Deflation offers the least client computation overhead as well as minimizes the communication overhead over the network. While

TABLE 3. ABLATION STUDY OF THE PROPOSED TECHNIQUES. THE SERVER-COMP. REFERS TO THE ADDITIONAL TIME SERVER WOULD REQUIRE TO FINISH ANY COMPUTATION ON THE HOMOMORPHICALLY ENCRYPTED DATA USING FPGA ACCELERATOR.

Technique	**Client Compute (Enc+Dec)** Time in ms+ms	**Comm. Overhead (Client→Server, Client←Server)** Data size in MB+MB	**Server-Comp.** Time in ms
Plain	204.3 + 41.2	28.0 + 0.98	0.0
Deflate-only	183.3 + 14.2	28.0 + 0.98	5.2
Boost-only	52.2 + 41.2	0.98 + 0.98	15.3
Naive Deflate +Boost	93.2 + 14.2	4.90 + 0.98	19.7
Our Boosted-Deflation	**28.2 + 14.2**	**0.98 + 0.98**	**15.1**

offloading the time with only one bootstrapping per homomorphic ciphertext to the server.

3.7. Comparison with Prior Techniques

The only other technique in the literature that aims to reduce the CoCo burden on the client and network is HHE [12]. This technique allows the user to encrypt via SE, and the server must perform decryption via a homomorphically encrypted key to obtain a homomorphic ciphertext (detailed in Appendix D). For comparisons, we first consider the HHE implementation [14] that uses the standard AES in stream modes (e.g., Counter and GCM) to help reduce computation depth. Stream mode helps prevent sequential bootstrappings by allowing them to be executed in parallel.

The benchmarks on HHE presented in [14] demonstrate that 87/105/123 bootstrapping operations are computed by the server for AES-128/192/256 decryption in stream mode using CKKS. Although our technique has effectively reduced communication overhead, it still amounts to $4\times$ the plaintext size. Consequently, while we experience $4\times$ greater communication overhead, we offload significantly less computation to the server (10 – $14\times$), as shown in Figure 8. In [14], the authors pack 1-bit per plaintext slot and utilize bit-slicing. This implies that for one AES-256 ciphertext of size 128-bit, 128 FHE ciphertexts are generated. The authors pack 2^{15} AES ciphertexts into 128 FHE ciphertexts. For instance, a plaintext of size 32B and another of 512KB will both result in 128 ciphertexts, each requiring 123 bootstrappings.

Let us consider the example of Homomorphic Logistic Regression Training [36], [37], which requires 96 bootstrapping operations. The existing HHE-based technique will offload 123 bootstrapping per 512KB plaintext. Thus, more time is spent in homomorphic decryption than in actual function evaluation. As the dataset size increases [48] ($\approx 47MB$ size testing-set), the number of bootstrapping for HHE-decryption increases to $11,562$ ($\frac{123 \cdot 47MB}{512KB}$). The required amount of bootstrapping operations in the DNN benchmark on [48] is $7,000$; with HHE, this increases by 256%. In contrast, our technique offloads up to 98% less (222-444) bootstrapping operations.

A similar conclusion can be drawn from the multiplicative depth consumption reported by newer, more efficient, non-standard HHE schemes [49], [50], for example, Pasta [7]. Pasta is one of the state-of-the-art efficient schemes that enables HHE. As detailed in Appendix E, Pasta's homomorphic symmetric decryption routine consumes $320\times$ more depth, and Pasta-3/4 require $32/16\times$ higher computation for encrypting the same amount of data (2^{15} data elements). Note that in HHE, only the encryption is symmetric. The final decryption on the client side is homomorphic. Thus, our proposed Boost technique can always be utilized to avoid the homomorphic decoding step. These schemes were initially proposed specifically for the BGV/BFV FHE schemes [26], [27], which are limited to process integers. In [13], [51], the authors, for the first time, demonstrated how to obtain a CKKS ciphertext using transciphering (bootstrapping) to convert a BGV ciphertext to a CKKS ciphertext. Therefore, an additional bootstrapping step is offloaded to the server along with homomorphic symmetric decryption to obtain the CKKS ciphertext. The ciphertext ratio for these techniques is also $1.54\times$, instead of 1 for AES.

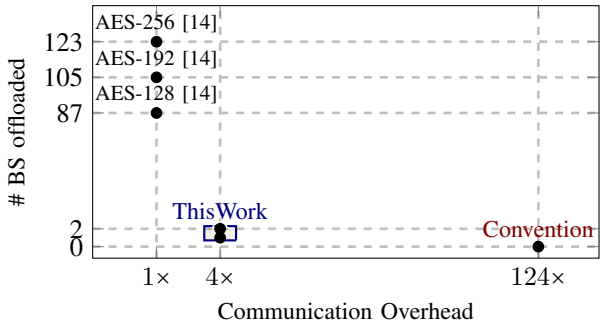

Figure 8. A comparison with works utilizing HHE to reduce CoCo. BS refers to Bootstrapping, for $L = 30$. Our work offloads at most two bootstrapping depending on the function.

Another prior work, [52], proposes an efficient ring-packing technique using Module-LWE ciphertexts at zero computational depth, which finds applications in transciphering. As the message in the ciphertext after packing is not encoded, the authors also utilize a HalfBoot technique, similar to the RtF framework in HERA [13], to gain computational depth after packing. The utilized ciphertext form is $\mathbf{c} \leftarrow \{a \cdot s + e, a\}$, where a seed can generate the a component on the server side instead of being sent via the communication channel, reducing the communication overhead. Note that this ciphertext structure differs significantly from the original structure proposed in CKKS/BGV ($a \cdot v + e$, where v and e are both unknown) and prevents on-the-fly computation of $\mathbf{c}_1$ component by the server. Therefore, while we can not consider this structure, such works can also utilize our design (section 4) to achieve acceleration. Furthermore, our encoding/decoding elimination analysis from enc/decryption remains useful.

4. Client Enc/Decryption Framework

This section details the Boosted-Deflation framework, which includes a detailed design exploration for software and hardware platforms, focusing on resource-constrained devices commonly used in client-side FHE.

4.1. Challenges

A common limitation of prior works [9], [16]–[18], [53] is their choice of parameters, which at most support a polynomial degree of 2^{14}. This degree is insufficient to enable *packed bootstrapping*.

① **Memory Constraints**. For 60-bit first modulus [18], storing one residue polynomial requires $2^{16} \times 60 = 0.5$MB of on-chip memory. Given the constraints of the platforms commonly used for client-side FHE, this is a considerable amount of memory. Therefore, it is crucial to minimize the number of polynomials needing on-chip storage at any instance during computation.

② **Off-Chip Communication.** Due to the constrained on-chip memory, constant data, such as the public keys, is stored off-chip. This results in expensive data transfers between off-chip and on-chip memory, which introduces additional communication overhead and negatively impacts the performance. Furthermore, data transfers on client-side FHE consume significant energy. Hence, running FHE on resource-constrained or battery-powered devices is challenging, especially for smartphones or IoT devices.

③ **Limited Application Support**. Although a majority of practical applications (e.g., ResNet Inference [54]) require bootstrapping, most of the prior client-side FHE works [9], [16], [19], [24] do not support parameters that could employ packed bootstrapping.

④ **Multi-Moduli Support**. To support encryption for all moduli ($q_0 \dots q_L$), generic Montgomery [55] or Barrett [56] reduction are utilized. These are expensive and involve several multiplications. In contrast, our Boosted-Deflation technique simplifies this by only using one fixed prime q_0 on the client side. Hence, a Mersenne prime can be chosen for a cheap add-shift-based reduction.

4.2. Framework Evaluation

❶ **Software Proof-of-Concept.** We first validated our proposed Boosted-Deflation technique via an implementation on the CPU. We utilize the existing HEAAN-library [57] and adapt it with minimal changes. HEAAN-library fully supports CKKS routines and uses $L = 28$, $\sigma = 3.2$, and $h = 64$, and $n = 128$. We employ HEAAN parameters for consistency, and our technique is not restricted to specific secret-key distributions or slot count. We run both the old and adapted versions on 12th Gen Intel® Core™ i7-1260P×16 Processor. The results show that baseline memory allocated to the program is approximately 233MB, and after encryption, this gets increased to 261.4MB. Our technique's overall memory consumption is only 233.26MB, thus reducing storage for ≈56 residue polynomials. Table 4 shows that the old version takes 204.3/41.2 milliseconds (ms) for encryption/decryption; it reduces to 28.2/14.2ms with our technique.

❷ **Analysis on Embedded Microcontrollers.** Next, we analyze the benefit of Boosted-Deflation on the Azure Sphere Cortex-A7 (A7) and the Nordic nRF52840 Cortex-M4 (M4) microcontrollers using the Embedded SEAL library [24]. The challenge here is that Embedded SEAL only provides benchmarks for ring degree $N = 2^{14}$, which is too low for bootstrapping. Therefore, we first estimate its runtime using existing processing capabilities for a higher ring degree. Since the bit-width of primes considered in Embedded SEAL is 30-bit, we also consider the same bit-width for fair comparisons, implying $L = 56$ for $N = 2^{16}$. Table 4 provides the estimated Embedded SEAL encryption runtime for both A7 and M4 as 12.35 and 58.46 seconds. We reduce it by up to 16×.

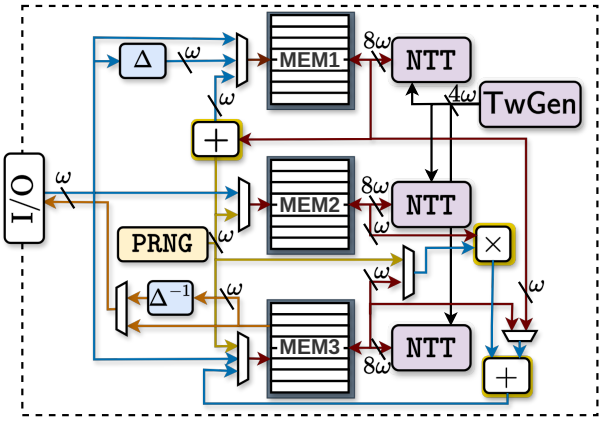

Figure 9. The proposed high-level architecture of the client device performing homomorphic encryption/decryption.

❸ **Hardware Design on FPGA.** We set our target as the low-cost Kintex-7 FPGA (Genesys2, xc7k325tffg900-2)[1], also utilized in state-of-the-art work [16]. This FPGA features 204k Look-Up-Tables (LUTs), 408k Flip-Flops (FFs), 840 Digital Signal Processors (DSPs), and 445 Block Random Access Memories (BRAMs) with 36kb capacity. Of the 445 BRAMs, 40 are occupied by the used MicroBlaze CPU. Thus, the available memory can store a maximum of three residue polynomials (each occupying 108 36kBRAMs) at 82.3% utilization. We used Xilinx Vivado 2022.2 to run synthesis, place & route and verified correctness by running the design on the FPGA. The design occupies 31.4k LUTs, 34.9k FFs, 204 DSPs, and 326 36k BRAMs. The runtime reports are present in Table 4, and the architecture is shown in Figure 9. A detailed design concept is presented in Appendix G, with utilization of the building blocks shown in Table 5.

❹ **ASIC Synthesis Assessment.** We also present ASIC-synthesis results using TSMC 28nm and ASAP7 [58] 7nm libraries to compare with prior ASIC works [9]. The synthesis is performed using Cadence Genus 2019.11, and clock-gating is employed to help reduce power consumption. The complete design consumes 3.36 mm^2 are on the 28nm process and 0.22 mm^2 area on the 7nm process. The SRAM memories alone consume 83% of the area (2.89 mm^2 in 28nm). The maximum total power consumption of the device is 2.79 watts. Encryption takes 0.70 ms, and decryption operations consume 0.49 ms. Thus, the chip would perform $1,429$ encryptions and $2,041$ decryptions per second.

1. Our implementation is available at https://github.com/aikata10/Client_Boost_Deflate

TABLE 4. COMPARISON ON SOFTWARE (CPU), EMBEDDED (MICRO-CONTROLLERS), AND HARDWARE (FPGA AND ASIC) PLATFORMS.

Work	Platform	Freq. (GHz)	$\log_2$(N,n)	$\log_2 \mathbf{q}_0$	BS	Area				Latency (ms)		
						LUT	FF	DSP	BRAM	NTT	Encr.	Decr.
HEAAN [57]	CPU 12thGen i7	4.7	16, 7	60	✓	-				0.95	204.3	41.2
This Work	CPU 12thGen i7	**4.7**	**16**, **7**	**60**	✓	-				**0.95**	**28.2**	**14.2**
Emb.SEAL [24]	CortexA7	0.49	16, 15	30‡	✓	-				106	12,354	-*
Emb. SEAL [24]	CortexM4	0.06	16, 15	30‡	✓	-				544	58,463	-*
This Work	CortexA7	**0.49**	**16**, 15	**30**	✓	-				**106**	**753**	-*
This Work	CortexM4	**0.06**	**16**, 15	**30**	✓	-				**544**	**4,535**	-*
[19]†	Virtex-7	0.2	10, 9	32	✗	77,000	-	952	325.5	4e-4	1e-3	8e-4
[17]	Zynq US+	0.15	13, 12	32	✗	-	-	-	-	-	7.79	-
ALOHA [16]	Kintex-7	0.20	13, 12	54	✗	20,728	17,647	100	82.5	0.27	1.87	0.87
[18]	AlveoU250	0.25	16, 15	60	✓	1,179,000	1,036,000	12,288	828.5	0.13	16.9	3.9
This Work	Kintex-7	**0.19**	**16** , 15	**60**	✓	**31,421**	**34,887**	**204**	**326**	**0.71**	**3.78**	**2.64**
RACE [53]	ASIC 12nm	1	13, 12	30	✗	$0.06\ mm^2$				-	110.3	19.1
RACE [53]	ASIC 12nm	1	14, 13	30	✗	$0.12\ mm^2$				-	328.9	51.4
RISE [9]	ASIC 12nm	1	13, 12	30	✗	$0.11\ mm^2$				-	20.0	19.0
RISE [9]	ASIC 12nm	1	14, 13	30	✗	$0.18\ mm^2$				-	41.6	37.3
This Work	ASIC 7nm	**1**	**16**, 15	**60**	✓	$\mathbf{0.22}\ mm^2$				0.13	**0.70**	**0.49**
This Work	ASIC 28nm	**1**	**16**, 15	**60**	✓	$\mathbf{3.36}\ mm^2$				0.13	**0.70**	**0.49**

‡ Balanced implementation results generalized for parameter choice. ; † Utilizes BFV scheme; * Embedded SEAL does not support decryption.

TABLE 5. COMPONENT-WISE FPGA AREA UTILIZATION AND PERFORMANCE. DESIGN DETAILS ARE PROVIDED IN APPENDIX G.

Component	Area				Lat. (kcc)
	k LUT	k FF	DSP	BRAM	
MEM (3×)	0.8	1.4	0	108	65.6
NTT (3×)	6.8	7.2	48	0	131
TW Gen.	6.6	8.7	48	2	
M-Ops.(+×)	0.9	1	12	0	65.6
PRNG	0.7	0.5	0	0	131
Scaling	0.7	0.3	0	0	65.6
Total	**32/16%**	**36/9%**	**204/24%**	**326/73%**	-

5. Comparison with Related Works

The prior works do not support arbitrary packing. Our work offers this by default due to the proposed Boosted-Deflation technique. Importantly, all the prior works only provide accelerated computation. Our technique additionally helps reduce communication overhead by up to 31× (97%) for the parameters used in [18]. Table 4 compares our area and performance results to state-of-the-art works for client-side operations in FHE across all platforms.

❶ **Software.** Compared to HEAAN [57], we achieve up to 7× speedup for the same parameters running on the same platform without any optimization effort. This speedup is attributed solely to our proposed Boosted-Deflation technique. Although the old implementation processes 29× (for L = 28) more data during encryption, we only see 7× speedup. This is due to the concurrent processing of multiple residues on the multithreaded CPU. Since our technique allows using one modulus q_0, we do not leverage thread-level parallelism for single encryptions. Instead, the Boosted-Deflation technique allows multiple independent encryptions simultaneously, thus delivering a higher throughput. Moreover, our technique will provide higher speedups on single-core CPUs or microcontrollers common in client-side FHE compared to the multithreaded case.

❷ **Microcontrollers.** On Cortex-A7, the estimated encryption latency is 12.35 seconds, whereas, with Boosted-Deflation, the latency is only 0.75 seconds. On Cortex-M4, the speedup using the Boosted-Deflation technique compared to Embedded SEAL is 12.9×. Note that the Cortex-M4 only provides a single-precision floating-point unit, although FFT requires double precision. Hence, the compiler emulates double-precision FP arithmetic using integer arithmetic. This leads to a significantly increased runtime for the FFT, as reported using the Embedded SEAL [24]. The Cortex-M4 shows up to 100× worse FFT performance than the Cortex-A7 with 7.7× lower clock frequency. Furthermore, the double-precision FFT consumes 11.5× more clock cycles than NTT (operating on integers) for the same data size. This highlights the benefit of omitting the floating-point-based encoding proposed by our Deflate technique on microcontrollers without FPU.

❸ **FPGA.** The authors in [17] propose an accelerator for resource-constrained devices. Only the encryption is performed in hardware, while the software does the encoding. In contrast, [16] performs the floating-point-based encoding on the FPGA. Compared to our work, both works [16], [17] only support 8× smaller polynomials, yet we outperform [17] by 2× while being 2× slower than [16]. This leads to a scaled performance improvement of 16× and 4× compared to [17] and [16], respectively. Compared to [16], we require 4× more BRAMs than [16] although we support 8× larger polynomials. This is an advantage of the Deflate technique, which omits the DFT and avoids the corresponding memory in hardware.

[18] presents an implementation of high-performance encryption and decryption operations. The authors target the Alveo U250 datacenter FPGA and propose a large-scale design. Considering the NTT performance, [18] outperforms our NTT by 5.5 × with the same parameter set. Yet, due to our Boosted-Deflation optimizations, we reduce the overall encryption latency by 4.5×. In addition, we utilize 38×, 30×, 60×, 2.6× fewer resources in terms of LUT, FF, DSP, and BRAM, respectively. This significantly lower resource utilization allows the deployment of our design on resource-constrained FPGAs, whereas [18] solely applies to high-end FPGAs.

The work in [19] presents a hardware-software co-design for the encryption/decryption in the BFV [27] scheme. They target 64× smaller polynomials and 2× smaller primes than our work and reports only hardware

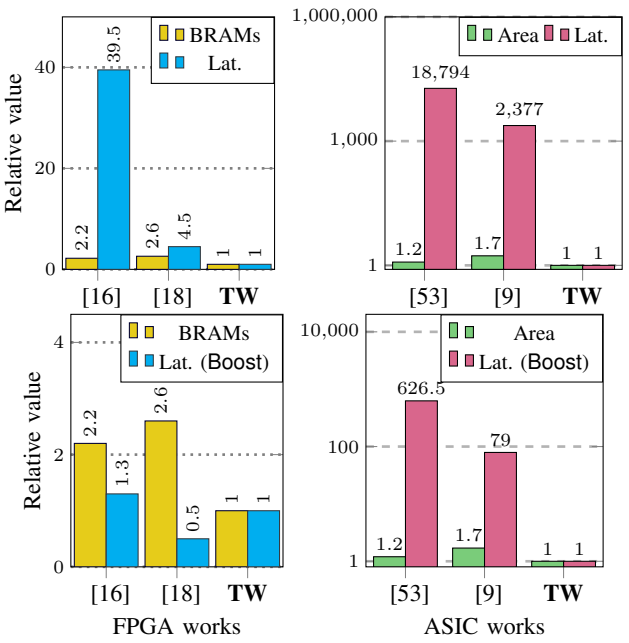

Figure 10. Comparison with prior works. The bottom plots apply Boost techniques on prior works.

runtime. We, therefore, scale their latency up to our parameter set. This indicates that we achieve 1.4× higher performance while requiring 2.5×, 4.9×, and 64× fewer LUTs, DSPs, and BRAMs, respectively. In addition, [19] targets a more powerful and larger FPGA- Virtex 7. This limits their applicability for resource-constrained client devices, highlighting the clear benefit of our methodology.

❹ **ASIC.** There are no standalone client-side FHE acceleration ASIC designs, as prior works [9] use SW for encoding and decoding steps. Naively comparing the latency shows that we deliver $30-157\times$ better performance. Although we achieve better results, the speedup comparisons provided are still unfair, as prior works also process fewer residue polynomials for smaller parameters. Thus, we normalize the area by scaling the memory with the difference in polynomial degrees and ω, since memories consume most of the area in the ASIC design and are the main constraints in FPGA. We present Figure 10 to compare our results using normalized area and performance metrics (for FPGA, we use BRAM area). The top two plots illustrate that our approach achieves the lowest area and runtime results, outperforming previous methods by more than $L+1 = 31$ times. The bottom two plots depict the scenario where prior works also have reduced computation overhead from the Boost technique, highlighting the advantage of our design exploration.

6. Application Benchmarking

We first perform an application benchmarking of video frame encryption [8] to show that our encryption performance and data overhead are at par with the available bandwidth in edge devices. This application is used in video surveillance systems to send encrypted video data to the cloud for processing [59]. It also covers the case of audio transmission required for [20]. A video is split into multiple frames, adaptable to available bandwidth and

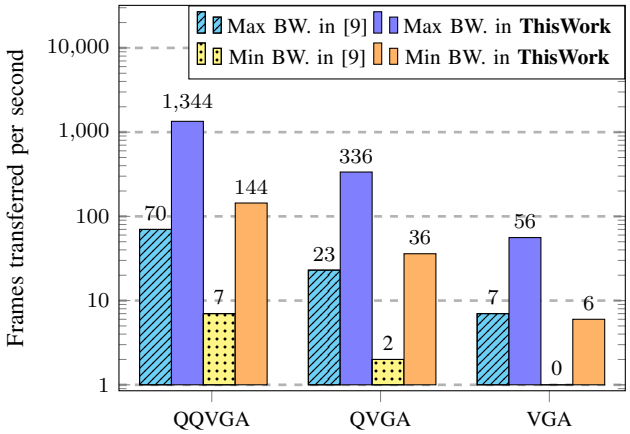

Figure 11. Frames transferred per second at max. (112.5 MBps) and min. (12.5 MBps) 5G available bandwidth (BW). The graphs are on a logarithmic scale.

desired resolution [60]. The resolution can range from VGA (640×480 pixels), QVGA (320×240 pixels), to QQVGA (160×120 pixels). The authors in [9] benchmark encrypting QVGA and QQVGA assuming grayscale pixels (8-bit) for bandwidth- 12.5 to 112.5 MBps. This bandwidth is for the network connecting the client's edge device to the cloud (e.g., mid-band 5G network [61]).

RISE [9] can encode one QQVGA frame per ciphertext ($N = 2^{14}, \log Q = 390$) and requires 3 such ciphertexts for a QVGA frame. One ciphertext size is 1.5MB ($2^{14} \cdot 2 \cdot 390$). Therefore, they can send 70 QQVGA frames per second when the maximum 5G bandwidth is available (112.5 MBps). It has also been reported that the baseline software implementation is unable to send even a single frame per second. Our results compared to [9] are reported in Figure 11. The ciphertext ($N = 2^{16}, \log q_0 = 60$) is only 0.9 MB in size ($2^{16} \cdot 2 \cdot 60$) due to the use of Boosted-Deflation technique. Consequently, we can send $1.7\times$ more ciphertext for the same bandwidth.

Additionally, due to the higher polynomial degree, we can pack $4\times$ more frames, for example, 12 QQVGA and 3 QVGA frames per ciphertext. Hence, we can send $8-20\times$ more frames per second, as shown in Figure 11. We also provide a benchmark for VGA resolution; for the minimal available 5G bandwidth, we can send six frames per second. The same could not be done using [9]. At the highest functioning rate of $1,344/12 = 112$ encryptions per second, a latency of 8.9ms per encryption is needed. Our readily available FPGA realized implementation (3.78ms) offers this. Note that the server has to process the video, and due to our bootstrappable parameter support, it can perform arbitrarily deep evaluations like deep neural networks [62]–[64].

6.1. Logistic Regression Training

The previous example shows that the proposed technique is at par for sending encrypted data to the server for practical applications like video or image analysis. Now, we will show an example of a logistic regression training benchmark, which would also put into perspective the CoCo overhead for the prior techniques versus our proposed Boosted-Deflation, as shown in Figure 12. The application does binary image classification between 3

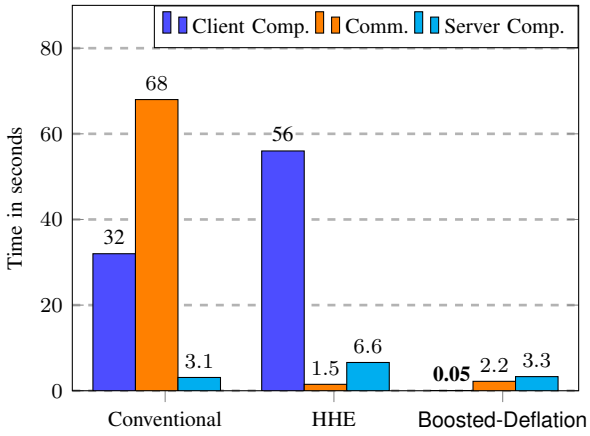

Figure 12. Time required for client and server computation, and communication for the Logistic Regression Training

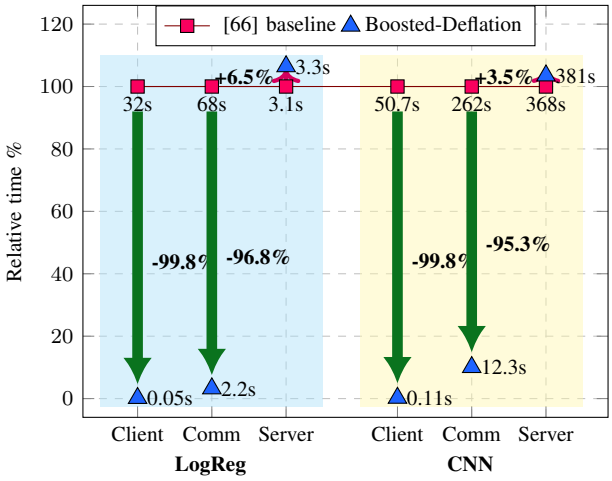

Figure 13. Relative runtime and communication breakdown of the secure-Logistic Regression Training and CNN-Inference pipeline

and 8 on the MNIST dataset [48], and its homomorphic evaluation was proposed in [36].

The initial size of each image was 28×28, which the authors preprocess and compress to 14×14 using the arithmetic mean for each 2×2 pixels. Out of the total of 13,966 samples in the dataset, 11,982 are used for training and the remaining for validation. The authors pack each pixel in one slot and use $N = 2^{16}$. Each ciphertext requires 8.8MB of storage. They use a block size 1024 and, therefore, require eight such ciphertexts per batch. Thus, overall, they require $12 \times 8 = 96$ ciphertexts for encrypting all the images and sending them to the server, which amounts to 845MB of data. The state-of-the-art client accelerators [9] require 32s to encrypt this huge amount of data. On the other hand, recent FHE hardware acceleration works for the server [47] show that this logistic regression training can be done in more practical 3.1s. It uses $n = 2^{15}$ and employs the non-sparse secret-key double-hoisting bootstrapping method proposed in [65].

Thus, we are bound to ask whether transferring such a huge amount of data is as fast as it can be computed. Sending 845MB of data, at the highest 5G available bandwidth, requires ≈8s; at the lowest bandwidth, this increases to ≈68s. Thus, on average, 38s are spent on sending data to the server alone. Now, if we use the prior technique of HHE to minimize this, the total ciphertext size will be 19MB ($196 \times 11,982 \times 64$), where each element is again encrypted in one slot of the integer HHE scheme [7], [13], [51]. This requires ≈1.5s; however, the computation of these many ciphertexts alone requires 56s [13], and it also offloads 2.1× larger computation overhead to the server, resulting in a total execution time of 6.6s.

With our proposed Boosted-Deflation technique and its framework, the ciphertext size reduces from 845MB to 76MB, and the server computation time merely increases from 3.1 to 3.8s, due to the additional overhead introduced by the Boosted-Deflation technique. Furthermore, we note that the value a pixel can take is in the range (0-7). Hence, 3 bits can be allocated for each pixel, and packing multiple pixels (3-6) in one slot can help reduce the ciphertext size to 13-25MB, which is very close to HHE. Furthermore, this also lowers the computation time on the server to 3.2-3.3s. With our proposed framework, the client-side encryption can also be done in 0.01-0.05s. The amount of time the client requires to encrypt such huge data is reasonable, given that ML training on high-end accelerators takes 3.2 seconds. Thus, the technique presents the best results in terms of minimising computation for both the client and server (99.8% reduction + 6.5% increase), as well as the communication overhead over the network (96.8% reduction), as shown in Figure 13.

6.2. CNN Inference

So far, we have established that the Boosted-Deflation technique outperforms the HHE technique in achieving the best CoCo balance. Next, we further evaluated our method on the open-sourced homomorphic 20-layer CNN (Convolutional Neural Networks) inference benchmark [66], to support its relevance. It uses the CIFAR-10/100 dataset [67], which consists of 10,000 images each of size $(32 \times 32 \times 3)$. One CKKS-ciphertext can pack 64 such images, as the authors use both real and imaginary slots. They use secret-key sparsity $h = 192$ to satisfy the security requirements, and one ciphertext occupies ≈ 22MB. Thus, it requires encrypting 157 ciphertexts and sending 3.2GB of data over the network.

With our technique, we reduce it to 0.98MB per ciphertext, and hence reduce the overall size to 154MB, which is 21× or 95.3% less, as shown in Figure 13. The reduction in encryption time is also reduced by 99.8% with our Boosted-Deflation framework, and the server overhead only increases by a mere 3.5%. As in the previous case, this overhead is due to the extra boostrapping required to support Boosted-Deflation.

7. Discussion

The proposed Boosted-Deflation technique is versatile and can be applied beyond plain FHE to other protocols that utilize FHE, such as Multi-Key FHE [68] and Multi-Party FHE [69], to help mitigate the CoCo overhead. These protocols frequently employ CKKS as well as other FHE schemes. The only limitation is the requirement for a single initial bootstrap. While negligible

for large ML workloads, it may affect shallow or low-depth applications.

■ **Applicability to other FHE schemes.** In this work, we used the CKKS [30] FHE scheme as a proof of concept. However, the same concepts can be applied to other RLWE-based FHE schemes such as BGV [26] and BFV [27]. These schemes are levelled and support bootstrapping to enable unlimited computation depth. In BGV/BFV bootstrapping, the procedure starts from a ciphertext at level q_ℓ (rather than q_0 as in CKKS).

The proposed Boost technique remains applicable since the encoding and decoding procedures correspond to public linear maps. This allows the client to perform encryption at a lower level, $q_{\ell'}$, with $0 < \ell' < \ell$, thereby reducing the number of levels required prior to bootstrapping. Similarly, the Deflate technique can still be applied by offloading the encoding/decoding operations to the server, since these transformations are public and rely on NTT-based polynomial transforms. The Boosted-Deflation approach can therefore be adapted to BGV/BFV-supported client implementations.

We note, however, that the practical gains in BGV/BFV are expected to be smaller compared to CKKS. In particular, CKKS involves floating-point DFT/IDFT-based encoding and decoding operations, which incur significant computational overhead. In contrast, BGV/BFV operate over an integer plaintext space, making the relative benefit of offloading smaller.

Interestingly, the encoding and decoding steps represent the main differences between CKKS and BGV/BFV encryption procedures. Once these steps are offloaded to the server, the proposed client-side implementation becomes largely scheme-agnostic across these RLWE-based constructions, providing additional flexibility for deployment and benchmarking.

Finally, we remark that the applicability of the proposed techniques to other FHE families such as TFHE [28] is not straightforward. TFHE is LWE-based, does not rely on packed ciphertext representations, and does not involve encode/decode transformations. Thus, the proposed techniques are not directly applicable in this setting. Additionally, TFHE exhibits significantly larger ciphertext expansion (e.g., ~8,000×) [70], which presents a different set of system design challenges and remains an interesting direction for future exploration.

■ **How can existing accelerators benefit from our technique?** Works utilizing the same polynomial degree [18] can directly implement our Boosted-Deflation technique to reduce computational demands. For other works, adaptations in storage requirements will be necessary. Figure 10 illustrates how applying the Boost technique can significantly reduce runtime in existing implementations. Many of these designs follow a strict pipelined flow with the DFT transform integrated as a part of the process. These designs can achieve substantial improvements by modifying the design methodology to fully incorporate the Boosted-Deflation technique. Implementing the Boost technique alone is sufficient to reduce memory overhead. Therefore, existing works can leverage the proposed technique for supporting bootstrappable parameters, which are beneficial for most FHE applications.

8. Conclusion

In this work, we aimed to design a framework for FHE clients and introduced Boosted-Deflation- a client-side FHE processing optimization technique. It addresses the significant computational and communication overhead on the client that has hindered the practical realization of FHE by minimizing the ciphertext expansion after encryption while also minimizing the server's computational overhead. The results show a 97% reduction in the computational and communication overhead. Moreover, the simplicity of our approach allows for straightforward deployment in existing as well as new frameworks.

Our framework features implementations and, hence, utility across multiple platforms- CPU, FPGA, and ASIC. The Boosted-Deflation technique reduces on-chip memory consumption and enables more efficient implementations. The results underscore the potential of our approach to make FHE a viable option for real-world applications, supporting secure and private computations without sacrificing performance on practical low-end client devices.

Acknowledgement

This work was supported in part by CONFIDENTIAL-6G EU project (Grant No: 101096435) and the State Government of Styria, Austria – Department Zukunftsfonds Steiermark.

References

[1] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, USA, 2009. [Online]. Available: https://searchworks.stanford.edu/view/8493082

[2] C. Gentry, "Computing arbitrary functions of encrypted data," *Commun. ACM*, vol. 53, no. 3, p. 97–105, Mar. 2010. [Online]. Available: https://doi.org/10.1145/1666420.1666444

[3] MindNetwork, "Fhe for web3." [Online]. Available: https://agent.mindnetwork.xyz/agenticworld

[4] C. F. Torres, F. Willi, and S. Shinde, "Is your wallet snitching on you? an analysis on the privacy implications of web3," in *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, J. A. Calandrino and C. Troncoso, Eds. USENIX Association, 2023, pp. 769–786. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/torres

[5] M. N. Islam and S. Kundu, "Poster abstract: Preserving iot privacy in sharing economy via smart contract," in *2018 IEEE/ACM Third International Conference on Internet-of-Things Design and Implementation, IoTDI 2018, Orlando, FL, USA, April 17-20, 2018*. IEEE Computer Society, 2018, pp. 296–297. [Online]. Available: https://doi.org/10.1109/IoTDI.2018.00047

[6] M. Naehrig, K. E. Lauter, and V. Vaikuntanathan, "Can homomorphic encryption be practical?" in *Proceedings of the 3rd ACM Cloud Computing Security Workshop, CCSW 2011, Chicago, IL, USA, October 21, 2011*, C. Cachin and T. Ristenpart, Eds. ACM, 2011, pp. 113–124. [Online]. Available: https://dl.acm.org/citation.cfm?id=2046682

[7] C. Dobraunig, L. Grassi, L. Helminger, C. Rechberger, M. Schofnegger, and R. Walch, "Pasta: A case for hybrid homomorphic encryption," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2023, no. 3, pp. 30–73, 2023. [Online]. Available: https://doi.org/10.46586/tches.v2023.i3.30-73

[8] M. A. Usman, M. R. Usman, and S. Y. Shin, "An intrusion oriented heuristic for efficient resource management in end-to-end wireless video surveillance systems," in *15th IEEE Annual Consumer Communications & Networking Conference, CCNC 2018, Las Vegas, NV, USA, January 12-15, 2018*. IEEE, 2018, pp. 1–6. [Online]. Available: https://doi.org/10.1109/CCNC.2018.8319177

[9] Z. Azad, G. Yang, R. Agrawal, D. Petrisko, M. B. Taylor, and A. Joshi, "RISE: RISC-V soc for en/decryption acceleration on the edge for homomorphic encryption," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 31, no. 10, pp. 1523–1536, 2023. [Online]. Available: https://doi.org/10.1109/TVLSI.2023.3288754

[10] S. Lage, F. Hörmann, F. Hanke, and M. Karl, "Collision risk analysis for LEO satellites with confidential orbital data," *CoRR*, vol. abs/2501.09397, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.09397

[11] J. H. Cheon, A. Kim, M. Kim, and Y. S. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, T. Takagi and T. Peyrin, Eds., vol. 10624. Springer, 2017, pp. 409–437. [Online]. Available: https://doi.org/10.1007/978-3-319-70694-8_15

[12] C. Gentry, S. Halevi, and N. P. Smart, "Homomorphic evaluation of the AES circuit," in *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, ser. Lecture Notes in Computer Science, R. Safavi-Naini and R. Canetti, Eds., vol. 7417. Springer, 2012, pp. 850–867. [Online]. Available: https://doi.org/10.1007/978-3-642-32009-5_49

[13] J. Cho, J. Ha, S. Kim, B. Lee, J. Lee, J. Lee, D. Moon, and H. Yoon, "Transciphering framework for approximate homomorphic encryption," in *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part III*, ser. Lecture Notes in Computer Science, M. Tibouchi and H. Wang, Eds., vol. 13092. Springer, 2021, pp. 640–669. [Online]. Available: https://doi.org/10.1007/978-3-030-92078-4_22

[14] E. Aharoni, N. Drucker, G. Ezov, E. Kushnir, H. Shaul, and O. Soceanu, "E2E near-standard and practical authenticated transciphering," *IACR Cryptol. ePrint Arch.*, p. 1040, 2023. [Online]. Available: https://eprint.iacr.org/2023/1040

[15] A. Bendoukha, A. Boudguiga, and R. Sirdey, "Revisiting stream-cipher-based homomorphic transciphering in the TFHE era," in *Foundations and Practice of Security - 14th International Symposium, FPS 2021, Paris, France, December 7-10, 2021, Revised Selected Papers*, ser. Lecture Notes in Computer Science, E. Aïmeur, M. Laurent, R. Yaich, B. Dupont, and J. García-Alfaro, Eds., vol. 13291. Springer, 2021, pp. 19–33. [Online]. Available: https://doi.org/10.1007/978-3-031-08147-7_2

[16] F. Krieger, F. Hirner, A. C. Mert, and S. S. Roy, "Aloha-he: A low-area hardware accelerator for client-side operations in homomorphic encryption," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[17] S. D. Matteo, M. L. Gerfo, and S. Saponara, "VLSI design and FPGA implementation of an NTT hardware accelerator for homomorphic seal-embedded library," *IEEE Access*, vol. 11, pp. 72 498–72 508, 2023. [Online]. Available: https://doi.org/10.1109/ACCESS.2023.3295245

[18] J. Lee, P. Duong-Ngoc, and H. Lee, "Configurable encryption and decryption architectures for ckks-based homomorphic encryption," *Sensors*, vol. 23, no. 17, p. 7389, 2023. [Online]. Available: https://doi.org/10.3390/s23177389

[19] A. C. Mert, E. Öztürk, and E. Savas, "Design and implementation of encryption/decryption architectures for BFV homomorphic encryption scheme," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 28, no. 2, pp. 353–362, 2020. [Online]. Available: https://doi.org/10.1109/TVLSI.2019.2943127

[20] E. H. Beni, L. Hoste, G. Heyman, P. Tsiaflakis, B. van Leeuwen, R. Geelen, M. Rivinius, and C. Bartoli, "Qrypt: End-to-end encrypted audio calls via blind audio mixing," *Real World Crypto*, 2025. [Online]. Available: https://www.youtube.com/live/6FQX2otSbuE?si=PMnUlIJT6m2Cuubp

[21] ATMEL, "8-bit microcontroller with 32kbytes in-system programmable flash," ATMEL, Tech. Rep., Accessed on July 9, 2024. [Online]. Available: https://www.microchip.com/en-us/product/atmega32

[22] ARM, "Arm cortex-m processor comparison table," ARM, Tech. Rep., Accessed on July 9, 2024. [Online]. Available: https://developer.arm.com/documentation/102787/latest/

[23] Aikata, A. C. Mert, S. Kwon, M. Deryabin, and S. S. Roy, "REED: chiplet-based scalable hardware accelerator for fully homomorphic encryption," *CoRR*, vol. abs/2308.02885, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2308.02885

[24] D. Natarajan and W. Dai, "Seal-embedded: A homomorphic encryption library for the internet of things," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 3, pp. 756–779, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i3.756-779

[25] H. L. Garner, "The Residue Number System," *IRE Trans. Electron. Comput.*, vol. 8, no. 2, pp. 140–147, 1959. [Online]. Available: https://doi.org/10.1109/TEC.1959.5219515

[26] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully homomorphic encryption without bootstrapping," *Electron. Colloquium Comput. Complex.*, p. 111, 2011. [Online]. Available: https://eccc.weizmann.ac.il/report/2011/111

[27] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 144, 2012. [Online]. Available: http://eprint.iacr.org/2012/144

[28] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[29] L. Ducas and D. Micciancio, "FHEW: bootstrapping homomorphic encryption in less than a second," in *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, ser. Lecture Notes in Computer Science, E. Oswald and M. Fischlin, Eds., vol. 9056. Springer, 2015, pp. 617–640. [Online]. Available: https://doi.org/10.1007/978-3-662-46800-5_24

[30] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "A full RNS variant of approximate homomorphic encryption," in *Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers*, ser. Lecture Notes in Computer Science, C. Cid and M. J. J. Jr., Eds., vol. 11349. Springer, 2018, pp. 347–368. [Online]. Available: https://doi.org/10.1007/978-3-030-10970-7_16

[31] D. Sprenkels, "The Kyber/Dilithium NTT," Accessed on July 21, 2023, https://dsprenkels.com/ntt.html.

[32] M. Scott, "A note on the implementation of the number theoretic transform," in *Cryptography and Coding - 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings*, ser. Lecture Notes in Computer Science, M. O'Neill, Ed., vol. 10655. Springer, 2017, pp. 247–258. [Online]. Available: https://doi.org/10.1007/978-3-319-71045-7_13

[33] S. Halevi and V. Shoup, "Faster homomorphic linear transformations in helib," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, 2018, pp. 93–120. [Online]. Available: https://doi.org/10.1007/978-3-319-96884-1_4

[34] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *J. Math. Cryptol.*, vol. 9, no. 3, pp. 169–203, 2015. [Online]. Available: http://www.degruyter.com/view/j/jmc.2015.9.issue-3/jmc-2015-0016/jmc-2015-0016.xml

[35] W. Liu and A. Nannarelli, "Power dissipation challenges in multicore floating-point units," in *21st IEEE International Conference on Application-specific Systems Architectures and Processors, ASAP 2010, Rennes, France, 7-9 July 2010*, F. Charot, F. Hannig, J. Teich, and C. Wolinski, Eds. IEEE Computer Society, 2010, pp. 257–264. [Online]. Available: https://doi.org/10.1109/ASAP.2010.5540986

[36] K. Han, S. Hong, J. H. Cheon, and D. Park, "Logistic regression on homomorphic encrypted data at scale," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium*

on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. AAAI Press, 2019, pp. 9466–9471. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33019466

[37] W. Jung, S. Kim, J. H. Ahn, J. H. Cheon, and Y. Lee, "Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus," *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, vol. 2021, no. 4, pp. 114–148, 2021. [Online]. Available: https://doi.org/10.46586/tches.v2021.i4.114-148

[38] N. Göttert, T. Feller, M. Schneider, J. Buchmann, and S. Huss, "On the design of hardware building blocks for modern lattice-based encryption schemes," in *Proceedings of the 14th International Conference on Cryptographic Hardware and Embedded Systems*, ser. CHES'12. Berlin, Heidelberg: Springer-Verlag, 2012, p. 512–529. [Online]. Available: https://doi.org/10.1007/978-3-642-33027-8_30

[39] M. R. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Laine, K. E. Lauter, S. Lokam, D. Micciancio, D. Moody, T. Morrison, A. Sahai, and V. Vaikuntanathan, "Homomorphic encryption standard," *IACR Cryptol. ePrint Arch.*, p. 939, 2019. [Online]. Available: https://eprint.iacr.org/2019/939

[40] Y. Bae, J. H. Cheon, J. Kim, and D. Stehlé, "Bootstrapping bits with CKKS," in *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part II*, ser. Lecture Notes in Computer Science, M. Joye and G. Leander, Eds., vol. 14652. Springer, 2024, pp. 94–123. [Online]. Available: https://doi.org/10.1007/978-3-031-58723-8_4

[41] A. A. Badawi and Y. Polyakov, "Demystifying bootstrapping in fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, p. 149, 2023. [Online]. Available: https://eprint.iacr.org/2023/149

[42] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for Approximate Homomorphic Encryption," in *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. B. Nielsen and V. Rijmen, Eds., vol. 10820. Springer, 2018, pp. 360–384. [Online]. Available: https://doi.org/10.1007/978-3-319-78381-9_14

[43] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, "OpenFHE: Open-Source Fully Homomorphic Encryption Library," in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, ser. WAHC'22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 53–63. [Online]. Available: https://doi.org/10.1145/3560827.3563379

[44] J. Bossuat, R. Cammarota, I. Chillotti, B. R. Curtis, W. Dai, H. Gong, E. Hales, D. Kim, B. Kumara, C. Lee, X. Lu, C. Maple, A. Pedrouzo-Ulloa, R. Player, Y. Polyakov, L. A. R. Lopez, Y. Song, and D. Yhee, "Security guidelines for implementing homomorphic encryption," *IACR Commun. Cryptol.*, vol. 1, no. 4, p. 26, 2024. [Online]. Available: https://doi.org/10.62056/anxra69p1

[45] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12696. Springer, 2021, pp. 648–677. [Online]. Available: https://doi.org/10.1007/978-3-030-77870-5_23

[46] J. Bossuat, A. Costache, C. Mouchet, L. Nürnberger, and J. R. Troncoso-Pastoriza, "Accurate and composable noise estimates for CKKS with application to exact HE computation," *IACR Commun. Cryptol.*, vol. 2, no. 2, p. 8, 2025. [Online]. Available: https://doi.org/10.62056/a3n5tx4e-

[47] R. Agrawal, L. de Castro, G. Yang, C. Juvekar, R. T. Yazicigil, A. P. Chandrakasan, V. Vaikuntanathan, and A. Joshi, "FAB: an fpga-based accelerator for bootstrappable fully homomorphic encryption," in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2023, Montreal, QC, Canada, February 25 - March 1, 2023*. IEEE, 2023, pp. 882–895. [Online]. Available: https://doi.org/10.1109/HPCA56546.2023.10070953

[48] Y. LeCun and C. Cortes, "MNIST handwritten digit database," http://yann.lecun.com/exdb/mnist/, 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[49] J. Ha, S. Kim, W. Choi, J. Lee, D. Moon, H. Yoon, and J. Cho, "Masta: An he-friendly cipher using modular arithmetic," *IEEE Access*, vol. 8, pp. 194 741–194 751, 2020. [Online]. Available: https://doi.org/10.1109/ACCESS.2020.3033564

[50] A. Canteaut, S. Carpov, C. Fontaine, T. Lepoint, M. Naya-Plasencia, P. Paillier, and R. Sirdey, "Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression," *J. Cryptol.*, vol. 31, no. 3, pp. 885–916, 2018. [Online]. Available: https://doi.org/10.1007/s00145-017-9273-9

[51] J. Ha, S. Kim, B. Lee, J. Lee, and M. Son, "Rubato: Noisy ciphers for approximate homomorphic encryption," in *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I*, ser. Lecture Notes in Computer Science, O. Dunkelman and S. Dziembowski, Eds., vol. 13275. Springer, 2022, pp. 581–610. [Online]. Available: https://doi.org/10.1007/978-3-031-06944-4_20

[52] Y. Bae, J. H. Cheon, J. Kim, J. H. Park, and D. Stehlé, "HERMES: efficient ring packing using MLWE ciphertexts and application to transciphering," in *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, H. Handschuh and A. Lysyanskaya, Eds., vol. 14084. Springer, 2023, pp. 37–69. [Online]. Available: https://doi.org/10.1007/978-3-031-38551-3_2

[53] Z. Azad, G. Yang, R. Agrawal, D. Petrisko, M. B. Taylor, and A. Joshi, "RACE: RISC-V soc for en/decryption acceleration on the edge for homomorphic computation," in *ISLPED '22: ACM/IEEE International Symposium on Low Power Electronics and Design, Boston, MA, USA, August 1 - 3, 2022*. ACM, 2022. [Online]. Available: https://doi.org/10.1145/3531437.3539725

[54] A. Ebel, K. Garimella, and B. Reagen, "Orion: A fully homomorphic encryption compiler for private deep neural network inference," *CoRR*, vol. abs/2311.03470, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2311.03470

[55] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, no. 170, pp. 519–521, 1985.

[56] P. Barrett, "Implementing the rivest shamir and adleman public key encryption algorithm on a standard digital signal processor," in *Annual International Cryptology Conference*, 1986. [Online]. Available: https://api.semanticscholar.org/CorpusID:19251441

[57] HEAAN and Cryptolab, "heaan, c++ interface to 'HEaaN'," HEaaN.stat, CryptoLab, Tech. Rep., Latest version updated in 2023. [Online]. Available: https://github.com/snucrypto/HEAAN?tab=readme-ov-file

[58] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm finFET predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X

[59] B. Lagesse, G. Nguyen, U. Goswami, and K. Wu, "You had to be there: Private video sharing for mobile phones using fully homomorphic encryption," in *19th IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events, PerCom Workshops 2021, Kassel, Germany, March 22-26, 2021*. IEEE, 2021, pp. 730–735. [Online]. Available: https://doi.org/10.1109/PerComWorkshops51409.2021.9431029

[60] V. V. Menon, A. Premkumar, P. T. Rajendran, A. Wieckowski, B. Bross, C. Timmerer, and D. Marpe, "Energy-efficient adaptive video streaming with latency-aware dynamic resolution encoding," in *Proceedings of the 3rd Mile-High Video Conference, MHV 2024, Denver, CO, USA, February 11-14, 2024*. ACM, 2024, pp. 21–27. [Online]. Available: https://doi.org/10.1145/3638036.3640801

[61] S. R. Thummaluru, M. Ameen, and R. K. Chaudhary, "Four-port mimo cognitive radio system for midband 5g applications," *IEEE Transactions on Antennas and Propagation*, vol. 67, no. 8, pp. 5634–5645, 2019.

[62] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems (video classification)," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/tutorials/video/video_classification

[63] S. Pentyala, R. Dowsley, and M. D. Cock, "Privacy-preserving video classification with convolutional neural networks," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 8487–8499. [Online]. Available: http://proceedings.mlr.press/v139/pentyala21a.html

[64] V. Sharma, M. Gupta, A. Kumar, and D. Mishra, "Video processing using deep learning techniques: A systematic literature review," *IEEE Access*, vol. 9, pp. 139 489–139 507, 2021. [Online]. Available: https://doi.org/10.1109/ACCESS.2021.3118541

[65] J. Bossuat, C. Mouchet, J. R. Troncoso-Pastoriza, and J. Hubaux, "Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys," in *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, ser. Lecture Notes in Computer Science, A. Canteaut and F. Standaert, Eds., vol. 12696. Springer, 2021, pp. 587–617. [Online]. Available: https://doi.org/10.1007/978-3-030-77870-5_21

[66] D. Kim and C. Guyot, "Optimized privacy-preserving CNN inference with fully homomorphic encryption," *IEEE Trans. Inf. Forensics Secur.*, vol. 18, pp. 2175–2187, 2023. [Online]. Available: https://doi.org/10.1109/TIFS.2023.3263631

[67] A. Krizhevsky, "Learning multiple layers of features from tiny images," pp. 32–33, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[68] T. Kim, H. Kwak, D. Lee, J. Seo, and Y. Song, "Asymptotically faster multi-key homomorphic encryption from homomorphic gadget decomposition," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, CCS 2023, Copenhagen, Denmark, November 26-30, 2023*, W. Meng, C. D. Jensen, C. Cremers, and E. Kirda, Eds. ACM, 2023, pp. 726–740. [Online]. Available: https://doi.org/10.1145/3576915.3623176

[69] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, "Multiparty computation with low communication, computation and interaction via threshold FHE," in *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings*, ser. Lecture Notes in Computer Science, D. Pointcheval and T. Johansson, Eds., vol. 7237. Springer, 2012, pp. 483–501. [Online]. Available: https://doi.org/10.1007/978-3-642-29011-4_29

[70] B. M. Case, S. Gao, G. Hu, and Q. Xu, "Fully homomorphic encryption with k-bit arithmetic operations," *IACR Cryptol. ePrint Arch.*, p. 521, 2019. [Online]. Available: https://eprint.iacr.org/2019/521

[71] H. Chen, I. Chillotti, and Y. Song, "Improved Bootstrapping for Approximate Homomorphic Encryption," in *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, ser. Lecture Notes in Computer Science, Y. Ishai and V. Rijmen, Eds., vol. 11477. Springer, 2019, pp. 34–54. [Online]. Available: https://doi.org/10.1007/978-3-030-17656-3_2

[72] M. Scott, "A note on the implementation of the number theoretic transform," in *Cryptography and Coding - 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings*. Springer, 2017, pp. 247–258.

[73] Aikata, A. C. Mert, M. Imran, S. Pagliarini, and S. S. Roy, "Kali: A crystal for post-quantum security using kyber and dilithium," *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 70, no. 2, pp. 747–758, 2023. [Online]. Available: https://doi.org/10.1109/TCSI.2022.3219555

[74] C. Bouillaguet, "Trivium," https://github.com/cbouilla/trivium, Dec. 2020.

Appendix

1. Bootstrapping

During homomorphic computations on the server, the multiplicative depth l gets successively reduced until it reaches $l = 0$. In this case, no further multiplications can be done, and a bootstrapping procedure (BS) [42], [65], [71] is required to refresh the computational depth of a ciphertext. However, this process consumes some of the refreshed depth, resulting in a ciphertext with a computational depth $l = L_{eff}$ $(< L)$ post-BS. Figure 3 provides an overview of the BS process used by CKKS [40], which consists of four steps.

1. Slot to Coefficient Conversion: After performing several computations, the initial ciphertext `c` encrypting m is left with 1 RNS-base q_0 $(l = 0)$. At this stage, the ciphertext contains an encryption of m in slot form. `SlotToCoeff` converts this to a ciphertext encrypting the polynomial form $m(x)$. This involves a homomorphic DFT computation achieved through matrix multiplication with the DFT matrix.

2. Modulus Raising: A `ModRaise` operation is performed to raise the modulus from q_0 to q_L. The resulting ciphertext $\mathtt{c}'$ now encrypts $m(x) + q_0 \cdot I(x)$. The extra term $q_0 \cdot I(x)$ has low-magnitude integer coefficients, multiplied by q_0. The next goal is to eliminate $q_0 \cdot I(x)$ to retrieve the original encryption by performing a modular reduction.

3. Coefficient to Slot Conversion: Now, the ciphertext is converted back to encrypt a vector in slot form $m + q_0 \cdot I$, again involving multiplication with the IDFT matrix.

4. Homomorphic Modular Reduction: The data, after `CoeffToSlot` described above, is prepared for `Homomorphic Modular Reduction`. This step usually uses a Chebyshev series-based polynomial approximation of the modular reduction function [43]. Together, these steps consume a computation depth $L_{boot} < L$, resulting in a ciphertext with a refreshed depth $L_{eff} = L - L_{boot}$.

2. Client-server data exchange

Table 6 gives the algorithmic description of the conventional client and server data exchange. The client-side computations are highlighted in detail. Algorithm 3 shows the iterative Cooley-Tukey algorithm for the NTT transformation.

3. Baby-step giant-step method

The matrix-vector multiplication required for processing the DFT transformation on the server side is expensive. This is not a problem on the client side, as data can be fetched and stored as desired. Contrarily, on the server side

TABLE 6. AN ALGORITHMIC DESCRIPTION OF THE CONVENTIONAL CLIENT AND SERVER COMPUTATIONS.

Client	Server
Encryption	
$(e_0, e_1)^*, v$ are refreshed every time	
$m' = \Delta \cdot \texttt{IDFT}(m) + e_0$	
$\boldsymbol{c_0} = \texttt{NTT}(v) \cdot \texttt{pk}_0 + \texttt{NTT}(m') \mod q_L$	
$\boldsymbol{c_1} = \texttt{NTT}(v) \cdot \texttt{pk}_1 + \texttt{NTT}(e_1) \mod q_L$	
Send $\texttt{c} = (\boldsymbol{c_0}, \boldsymbol{c_1})$ $\rightarrow$	$\rightarrow$ Receive $\texttt{c}$
Function Evaluation	
	$\acute{\texttt{c}} = f(\texttt{c})$
Receive $\acute{\texttt{c}}$ $\leftarrow$	$\leftarrow$ Send $\acute{\texttt{c}}$
Decryption	
$m' = \texttt{DFT}(\Delta^{-1}(\texttt{INTT}(\acute{\boldsymbol{c}}_1 \cdot \texttt{sk} + \acute{\boldsymbol{c}}_0)))$	

* Although we had mentioned that the error is refreshed every time, for clarity, we have given it two variable names e_0, e_1.

Algorithm 3 The Cooley-Tukey NTT algorithm [72]

In: A vector $\boldsymbol{x} = [x_0, \cdots, x_n - 1]$ where $x_i \in \mathbb{Z}_p$, $n, q \in \mathbb{Z}_q$.
In: Table of $2n^{th}$ roots of unity $\boldsymbol{g}$, in bit reversed order.
Out $\hat{\boldsymbol{x}} \leftarrow NTT(\boldsymbol{x})$, $x_i \in \mathbb{Z}_q$, in bit-reversed order.

$t, m \leftarrow (n/2), 1$
while $(m < n)$ **do**
  $k \leftarrow 0$
  for $(i = 0;\ i < m;\ i = i + 1)$ **do**
    for $(j = k;\ j < (k + l);\ j = j + 1)$ **do**
      $V \leftarrow \boldsymbol{x}[j + t] \times \boldsymbol{g}[m + i] \pmod q$ ▷ Butterfly Operation.
      $\boldsymbol{x}[j + t] \leftarrow \boldsymbol{x}[j] - V \pmod q$ ▷ Butterfly Operation.
      $\boldsymbol{x}[j] \leftarrow \boldsymbol{x}[j] + V \pmod q$ ▷ Butterfly Operation.
    end
    $k \leftarrow k + 2t$
  end
  $t, m \leftarrow t/2, 2m$
end
return $\boldsymbol{x}$

data vector is encoded/encrypted into polynomials, which do not offer much flexibility and consume significant multiplicative depth. To reduce this, the baby-step giant-step method [33] is used for expediting matrix-vector multiplication on encoded/encrypted data, as shown in the equation below.

$$\texttt{c}' = \sum_{k=0}^{N_2-1} \texttt{rot}_{kN_2} \left(\sum_{j=0}^{N_1-1} \texttt{diag}_{kN_1+j}(\mathbf{M}) \odot \texttt{rot}_j(\texttt{c}) \right)$$

where matrix ($\mathbf{M}$) represents encoded plaintext data for DFT transformation in this context, and the vector corresponds to encrypted ciphertext ($\texttt{c}$). N is the ciphertext size, and $N = N_1 \cdot N_2$. The $\texttt{rot}_j$ function rotates the input vector by j, and the $\texttt{diag}$ function gives rotated diagonals of $\mathbf{M}$.

The method involves three main steps: first, the diagonals of the matrix are homomorphically encoded; second, these encoded diagonals are rotated; and third, the rotated diagonals are multiplied with the rotated ciphertext, followed by additional rotations and accumulations.

4. Hybrid Homomorphic Encrytpion

HHE introduces an approach to mitigate the communication and computational overhead associated with traditional FHE. By incorporating a symmetric key-based encryption method, HHE streamlines the encryption process. The protocol involves two key phases: initialization and encryption using SE. During initialization, the client generates a symmetric key, encrypts it with the public key for FHE, and sends it to the server. Subsequently, the client encrypts data using SE and transmits it to the server using a nonce.

The server then performs homomorphic decryption on the SE ciphertext, resulting in a homomorphically encrypted ciphertext under the asymmetric (FHE) key, allowing further processing or storage. Throughout the process, various cryptographic operations, including encryption, decryption, and evaluation of homomorphic circuits, are employed to maintain data confidentiality and integrity. The transformation of ciphertext from one encryption form to another, termed transciphering, facilitates secure data transmission and computation in the cloud environment.

Researchers have addressed the limitations of traditional SE schemes, which primarily operate on Boolean data, by developing schemes compatible with real or integer plaintexts. For example, schemes like PASTA [7], HERA [13], and RUBATO [51] operate directly over prime fields and enable efficient encryption and communication while facilitating arithmetic operation for FHE schemes like BGV, BFV, and CKKS.

5. Comparison of Boosted-Deflation with HHE

We present the comparison with respect to one of the state-of-the-art schemes over integers- Pasta [7]. Pasta-4 requires up to 5 multiplications per encryption and can handle up to 32 data elements. However, the duplicated packing necessary for the utilized matrix multiplication technique hinders the efficient packed multiplication of multiple ciphertexts. Consequently, for 2^{15} data elements, the depth consumption would reach $5,120$ (with 1024 parallel executions). In contrast, our technique, which involves only one bootstrapping operation, would at most consume a depth of 16 for 2^{15} data elements.

In Pasta-3, the state comprises 256 elements, split into two parts of 128 each. The number of rounds is three, and each round involves a 128×128 invertible matrix generation and multiplication. The multiplicative complexity of both operations is the same- $128 \cdot 128$, and eight such operations are required. This brings the total multiplication cost to 2^{18}. It is interesting to note that, while FHE encrypts 2^{15} elements in 2^{16} degree polynomials, Pasta-3 can only encrypt $128 = 2^7$. Therefore, although Pasta-3 can encrypt 32 elements fast, it will need 2^8 more encryptions to encrypt 2^{15} elements, thus requiring total 2^{26} modular multiplications. For Pasta-4, the required computations for encrypting the same amount of data are halved due to half state size, which has a quadratic impact on multiplication count and a linear on the amount of data packed.

The polynomial degree in our work is $N = 2^{16}$, and for polynomial multiplication (the *most expensive* routine), a

Number Theoretic Transform (NTT) based technique is utilized. Each NTT transform requires $\frac{N \cdot \log N}{2}$ multiplications. Conventionally, this process would have repeated $31\times$, bringing the cost to $\approx 2^{26}$ modular multiplications, similar to that required for Pasta-3 and half of what is required for Pasta-4. With our proposed approach, the total number of multiplications required is $\approx 2^{21}$, which is $32\times$ less than those required for Pasta-3. A similar analysis for Pasta-4 shows that our proposed technique delivers $16\times$ less modular multiplication computation.

6. Algorithmic description of the proposed technique and Security Aspects

Algorithm 4 and Algorithm 5 provide algorithmic descriptions of FHE encryption and decryption routines after applying the proposed Boosted-Deflation technique.

The security definitions for the proposed Boosted-Deflation technique are as follows.

1. Encoding and decoding do not involve secret key routines. Thus, offloading this operation on encrypted data does not impact confidentiality guarantees.

2. Integrity: Once encrypted, data retains its security level regardless of whether it was originally encoded using DFT or any other transformation.

3. Attack Resilience: Since encoding and decoding are a publicly known transformation, an adversary gains no advantage by observing encoded messages before encryption.

Algorithm 4 New CKKS Enc.

In: $m \in \mathcal{R}$, $\mathtt{pk} = (\boldsymbol{pk}_0, \boldsymbol{pk}_1) \in \mathcal{R}^2_{q_0}$
Out: $\mathtt{c} = \text{Encrypt}(m) \in \mathcal{R}^2_{q_0}$

1: $v \xleftarrow{\$} \chi_{enc}$, $e_0, e_1 \xleftarrow{\$} \chi_{err}$
2: $\acute{m} \leftarrow \Delta \cdot m + e_0$
3: $v' \leftarrow \mathtt{NTT}(v) \bmod q_i$
4: $c_0^0 \leftarrow (pk_0^0 \cdot v' + \acute{m}_{\mathtt{NTT}})_{q_0}$
5: $c_1^0 \leftarrow (pk_1^0 \cdot v' + e_{1_{\mathtt{NTT}}})_{q_0}$
6: **return** $\mathtt{c} = \{\boldsymbol{c}_0, \boldsymbol{c}_1\} \in \mathcal{R}^2_{q_0}$

Algorithm 5 New CKKS Dec.

In: $\mathtt{c}, \mathtt{sk} \in \mathcal{R}^2_{q_0}$
Out: $m' = \text{Decrypt}(\mathtt{c})$

1: $c' \leftarrow \mathtt{INTT}(\langle \mathtt{c}, \mathtt{sk} \rangle)_{q_0}$
2: $m' \leftarrow \Delta^{-1} \cdot c'$
3: **return** $m' \in \mathcal{R}$

4. RLWE Security: The difficulty of solving RLWE instances remains unchanged as long as noise levels do not exceed decryption failure thresholds. Our approach follows standard parameter recommendations to ensure this condition holds.

5. Noise Management: Reducing the initial ciphertext modulus does not accelerate noise growth beyond what is already accounted for in standard FHE parameter selection.

6. Bootstrapping Resilience: The bootstrapping process inherently maintains security under the RLWE assumption by resetting noise accumulation. Additionally, noise flooding during bootstrapping may offer additional protection against IND-CPA^D attacks [45].

7. Overview of Framework Components

We present this overview in the context of hardware design, as for the software we adapt the exiting open source library HEAAN [57]. A key component of client-side FHE acceleration architecture is the NTT unit, which utilizes modular arithmetic. For converting the plaintext from floating-point representation to integer representation and vice versa during encode/decode, a scaling unit is also integrated which performs simple power-of-two multiplications of Δ/Δ^{-1}. The design details of the NTT and modular reduction unit are as follows.

7.1. NTT Unit. In our design, we use three parallel NTT cores. Algorithm 3 in the Appendix B details the NTT transformation. Each core performs one $N = 2^{16}$-point NTT on independent polynomials. This is used to compute $\mathtt{NTT}(m + e_0)$, $\mathtt{NTT}(e_1)$, and $\mathtt{NTT}(v)$ simultaneously during the encryption procedure. The implementation of NTT (shown in Figure 15) utilizes a divide-and-conquer approach by recursively calling the Butterfly Unit (BFU). Since we need both NTT and INTT transforms, our BFU is unified and performs both operations [73]. We further reduce the latency of NTT $4\times$ by instantiating four $BFUs$ per NTT core. This improved NTT latency matches the latency of other hardware components, such as PRNG. Hence, the overall encryption latency is balanced across all suboperations.

During each NTT transformation, a total of N twiddle factors (generated using $2N$-th primitive root of unity) are required. However, storing this amount of twiddle factors on-chip would require 108 BRAMs, which exceeds the available resources for the target FPGA. Therefore, we use on-the-fly generated twiddle factors to lower the BRAM consumption to just 2 BRAMs for storing initial twiddle factors. Since all three NTT cores perform the same operation, they require the same twiddle factors. Hence, we only instantiate one such twiddle factor generation module. This measure effectively reduces the resource and energy consumption.

7.2. Butterfly Unit Utilized in NTT/INTT. Each BFU performs dyadic addition, subtraction, and multiplication on the two input coefficients, with results reduced modulo q_0. The Gentleman-Sande (GS) [32] butterfly is utilized in Decimation-in-Frequency (DIF) transformations, while the Cooley-Tukey (CT) [32] butterfly is applied in Decimation-in-Time (DIT) transformations. Each butterfly operation takes two coefficients as input and produces two outputs. Additionally, the twiddle factor, ψ, is employed during butterfly computations [32]. Consequently, CT and GS butterfly configurations are used for the NTT and INTT. To facilitate both NTT and INTT, a unified butterfly core is designed to accommodate both CT and GS butterfly configurations.

7.3. Modular Reduction Unit and Data Flow. The modular reduction in prime field multiplication is a central hardware design aspect. As discussed earlier, we only need to support one modulus. To leverage this fact, we choose the 60-bit prime $q_0 = 2^{60} - 2^{18} + 1$ and design an optimized add-shift-based reduction unit shown in Figure 16. Compared to classical word-level Montgomery reduction, our approach uses 32% less LUTs and no DSPs. In addition, our reduction unit's power consumption is $3.5\times$ lower.

7.4. High-level architecture. The algorithm described in Table 2 serves as a reference for discussing architectural

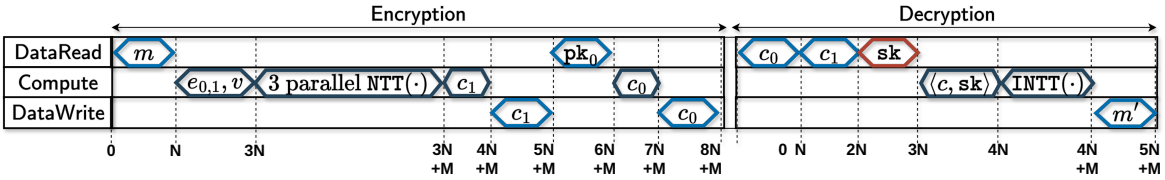

Figure 14. The proposed timeline of the encryption-decryption routines such that maximum residue polynomial storage required at any instance is three. Here, M is the latency of one NTT (or three parallel NTTs).

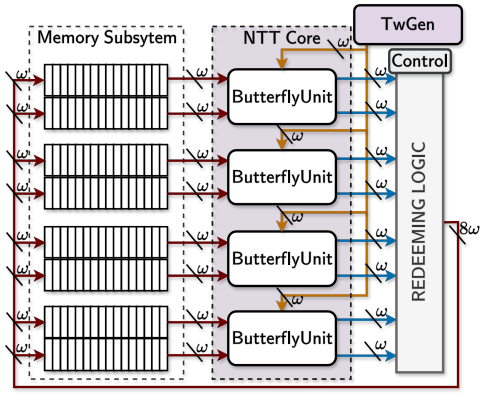

Figure 15. The NTT architecture which utilizes four butterfly units.

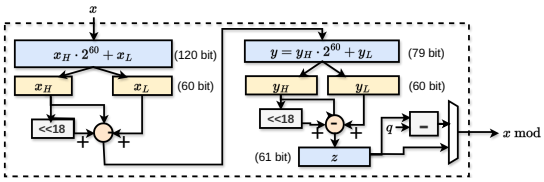

Figure 16. The shift-add based Modular Reduction unit.

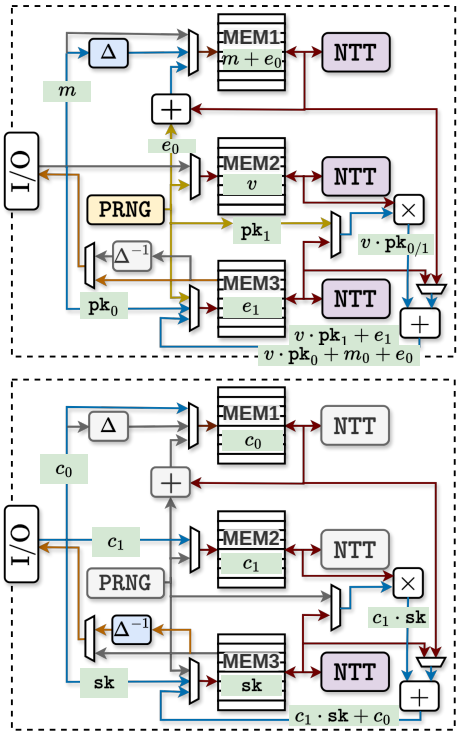

Figure 17. The data flow during encryption and decryption routines in the proposed high-level architecture.

design techniques. Our proposed architecture is shown in Figure 9. The central components are the three memories MEM1 to MEM3, wherein each memory stores one $N = 2^{16}$-degree residue polynomial. Attached to the memories are three unified NTT cores capable of performing forward NTT during encryption on up to three polynomials simultaneously and an inverse NTT during decryption. In addition, modular arithmetic modules $(\times, +)$ are integrated. These modules perform coefficient-wise arithmetic between two polynomials in the NTT domain. During encryption, a lightweight Pseudo-Random Number Generator (PRNG)- Trivium [74] is followed by sampling to generate v, e_0, and e_1 directly in hardware. In addition, one part of the public key, $pk_1 \in \mathcal{R}_{q_0}$, is also sampled on-demand based on a given seed. These aspects address reducing off-chip communication effort to minimize the overall latency. The I/O interface of our hardware accelerator facilitates control and data exchange with the DDR memory. To stream a large amount of data to and from the DDR, a Direct Memory Access (DMA) controller is utilized. This DMA transfers one coefficient size (ω) per clock cycle in our design.

7.5. Data Flow. The data flow for the complete encryption and decryption procedure is illustrated with the timeline in Figure 14. An architecture-based flow for encryption decryption routines is depicted in Appendix H.

During encryption, the client only needs to send the message m and public key $pk_0 \in \mathcal{R}_{q_0}$. This data is sent once, and all subsequent data is generated dynamically (on-the-fly). First, the message is received, followed by the generation of the error $(e_0,\ e_1)$ and v polynomials. These fully occupy the three available memory blocks, preventing additional data from being loaded and at this stage, only the c_1 component of the ciphertext can be generated. Hence, initially, the data in all three memories $(m + e_0,\ v,$ and $e_1)$ is processed via three parallel NTT cores. Following this, the data in the second and third memories $(\texttt{NTT}(v),\ \texttt{NTT}(e_1))$ is processed along with on-the-fly generated pk_1 to compute c_1. Finally, c_1 is offloaded to the client device, freeing one memory block. This allows pk_0 to be loaded into this memory block to compute c_0.

As depicted in the figure, the latency of error sampling is $2N$, whereas all other linear operations take N clock cycles. The latency of the NTT is $M = \frac{N \log_2 N}{2 \cdot \#BFU}$, where $\#BFU$ stands for the number of butterfly units [32] utilized per NTT core. As the maximum latency among the non-NTT steps is $2N$, we have chosen $\#BFU = 4$ so that for $\log_2 N = 16$, $M = 2N$ and the NTT unit's

latency is at par with the remaining units. This flow can be pipelined for larger, non-constrained environments. Consequently, the encryption and decryption routines will offer the same latency but much higher throughput. The complete data flow for encryption with the architecture is shown in Figure 17 (Appendix H).

For decryption, the two residue polynomials in the ciphertext (c_0, c_1) and the secret key $\mathtt{sk}$ are initially loaded into the three available memory blocks. After a coefficient-wise multiplication and addition ($\langle c_1, \mathtt{sk} \rangle + c_0$), one of the NTT units is utilized to perform the INTT operation (shown in Figure 17, Appendix H). Following this, the resultant plaintext is sent back to the DDR memory.

8. Operation-wise Data Flow

The flow diagram in Figure 17 illustrates the flow of plaintext, public key and ciphertext during the CKKS encryption procedure. Similarly, the second flow diagram in Figure 17 shows the ciphertext, secret key, and resultant plaintext flow during the CKKS decryption procedure.