# Silenzio: Secure Non-Interactive Outsourced MLP Training

Jonas Sander
University of Luebeck
j.sander@uni-luebeck.de

Thomas Eisenbarth
University of Luebeck
thomas.eisenbarth@uni-luebeck.de

***Abstract*—Outsourcing ML training to cloud-service-providers presents a compelling opportunity for resource constrained clients, while it simultaneously bears inherent privacy risks. We introduce Silenzio, the first fully non-interactive outsourcing scheme for the training of MLPs that achieves 128 bit security using FHE (precisely TFHE). Unlike traditional MPC-based protocols that necessitate interactive communication between the client and server(s) or non-collusion assumptions among multiple servers, Silenzio enables the "fire-and-forget" paradigm without such assumptions. In this approach, the client encrypts the training data once, and the server performs the training without any further interaction.**

**Silenzio operates entirely over low-bitwidth integer to mitigate the computational overhead inherent to FHE. Our approach features a novel low-bitwidth matrix multiplication gadget that leverages input-dependent residue number systems, ensuring that no intermediate value overflows 8 bit. Starting from an RNS-to-MRNS conversion process, we propose an efficient block-scaling mechanism, which approximately shifts encrypted tensor values to their user-specified most significant bits. To instantiate the backpropagation of the error, Silenzio introduces a low-bitwidth gradient computation for the cross-entropy loss.**

**We evaluate Silenzio on standard MLP training tasks regarding runtime as well as model performance and achieve similar classification accuracy as MLPs trained using PyTorch with 32 bit floating-point computations. Our open-source implementation of Silenzio represents a significant advancement in privacy-preserving ML, providing a new baseline for secure and non-interactive outsourced MLP training.**

## I. Introduction

Machine learning (ML) and particularly the rise of powerful artificial neural networks (NNs), has transformed numerous industries through significant breakthroughs in pattern recognition, decision-making, and predictive analytics. Among these, Multi-Layer Perceptrons (MLPs) constitute a foundational and versatile class of NNs, widely employed in critical areas ranging from medical diagnostics to financial analytics. However, the ever-increasing complexity of ML models and size of datasets necessitate substantial computational resources that are often unavailable to resource-constrained clients, such as individual users (e.g., IoT or medical devices) or small-scale enterprises. Consequently, outsourcing ML training tasks to cloud-service-providers emerges as a practical solution.

Despite the clear advantages of outsourced NN training, serious concerns regarding privacy and security arise, especially when the underlying datasets contain sensitive personal or economically valuable information. Traditional solutions have predominantly utilized Multi-Party Computation (MPC)-based protocols, e.g., [1]–[12], relying heavily on interactive communication or strict non-collusion assumptions among multiple servers. Such requirements introduce significant practical limitations, hindering their widespread adoption in scenarios demanding secure, seamless and scalable solutions.

Existing non-interactive schemes for NN training not requiring non-collusion assumptions [13], [14] solely rely on Fully Homomorphic Encryption (FHE), but only guarantee around 80 bits of security in their evaluation to significantly reduce their computational overhead and are therefore considered insecure by today's standards. Furthermore, there is no open-source implementation of such schemes, massively hindering further advancements in the development of practical and secure solutions for non-interactive outsourcing of ML training.

To address the need for secure and fully non-interactive outsourced training of MLPs, we introduce Silenzio. Silenzio solely relies on FHE — particularly Fast Fully Homomorphic Encryption over the Torus (TFHE) [15] — and provides a robust security guarantee of 128 bits. To meet performance requirements and minimize the computational overhead, Silenzio draws from recent advancements in hardware-accelerated ML [16] and shows how similar approaches can accelerate TFHE-protected ML training. Silenzio leverages exclusively low-bitwidth integer arithmetic, never exceeding 8 bits, thus significantly mitigating the computational overhead characteristic of FHE-protected computations. As part of Silenzio, we propose three new building blocks to enable effective training without exceeding the 8 bit limit for all FHE computations. First, we introduce a low-bitwidth matrix multiplication gadget leveraging input-dependent residue number systems (RNS) and a low-bitwidth modular summation routine. Based on a new vectorized and FHE-protected implementation for RNS to mixed-radix number system (MRNS) conversions, we propose the second building block: a novel block-scaling gadget that approximately shifts the values of an encrypted input tensor given in RNS representation to its user specified $\Gamma$ most significant bits. As the third and final building block, we propose a simple approximated gradient computation for the cross-entropy loss compatible with TFHE.

Our implementation of Silenzio is based on Zama's state-of-the-art Concrete library [17]. We comprehensively evaluate Silenzio using standard benchmark datasets and various MLP configurations, establishing a new performance baseline for privacy-preserving outsourced MLP training. Furthermore, SILENZIO will be released as open-source, aiming to encourage further research regarding the non-interactive outsourcing of NN training.

### A. Contributions

We introduce SILENZIO, the first fully non-interactive training scheme for MLPs that provides full-strength security while achieving unprecedented performance due to the following contributions:

- SILENZIO achieves 128 bit of security while only requiring low-bitwidth integer computations and never exceeding 8 bit for FHE-computations.
- A new low-bitwidth matrix-multiplication gadget for up to 8 bit input-matrices that leverages input-dependent residue number systems, a low-bitwidth summation routine, and optionally a Karatsuba-inspired multiplication engine to keep all FHE-processed values, including the output, within 8 bit value ranges.
- A new FHE-protected (block-)scaling gadget, Shift2MSBs$^{\pm}$, that approximately shifts the values of an input-tensor represented in a residue number system to its user-specified $\Gamma$ most significant bits.
- As part of Shift2MSBs$^{\pm}$, we provide a fast FHE-protected implementation for number conversions from residue number systems to associated mixed-radix number systems.
- A low-bitwidth and TFHE-friendly gradient computation for the cross-entropy loss.
- SILENZIO's end-to-end training approach is implemented using the state-of-the-art Concrete library, providing good extensibility and a low-barrier starting point for further research. To the best of our knowledge, we provide the first open-source implementation of a non-interactive (without non-collusion assumption) cryptographic outsourcing scheme for MLP training.

We will release the code upon acceptance.

## II. PRELIMINARIES

We note vectors with bold lower-case (e.g., $\mathbf{m}$) and matrices respective tensors with bold upper-case letters (e.g., $\mathbf{W}$). $\mathbb{Z}_q$ denotes the ring of integer modulo $q$ and $\lceil\cdot\rceil$ rounding upwards to the next integer. We summarize used notations in Table I.

### A. Non-Interactive Outsourcing

Cryptographic neural network computations are broadly categorized into four distinct scenarios, each differing substantially in terms of interactivity, assumptions, and practical constraints (see also [18]). To further motivate SILENZIO's scenario and point out the differences to other settings, we shortly introduce them in the following.

*Oblivious Inference/Training* schemes typically involve a single server providing inference or training services without learning the client's input data [19]–[33]. Although effective for protecting inputs, these schemes inherently rely on frequent client-server interactions beyond initial setup, leading to significant computational and communication overheads, often outweighing the efficiency benefits of cloud outsourcing. *Private Inference/Training* addresses scenarios involving multiple data owners who collaboratively compute inference results or train models without mutually revealing sensitive input data [3], [7], [10], [34]–[45]. However, these approaches generally require substantial inter-party communication, making them unsuitable for clients seeking minimal online involvement.

To alleviate these communication-intensive requirements, *Semi-Non-Interactive Outsourced Inference/Training* schemes like e.g., [1]–[12], [45] adopt a more cloud-compatible "fire-and-forget" paradigm. In these protocols, the client initially secret-shares their input among multiple independent servers, after which they may go offline until computation completion. This approach significantly reduces online communication but introduces critical security assumptions: the client must fully trust that servers do not collude, which might be difficult to guarantee — particularly when servers belong to the same organization or jurisdiction. To eliminate reliance on potentially unrealistic non-collusion assumptions, SILENZIO leverages the *Non-Interactive Outsourced Inference/Training* setting (for an overview, see Table II and Section VI). Here, the client encrypts their data using cryptographic primitives — such as FHE — and sends the ciphertext to a single server. After initial transmission, no further interaction between the client and server is required until the server completes its computation. This setting entirely removes the need for non-collusion assumptions, making it a straightforward "drop-in replacement" for conventional, unprotected cloud offerings. In some sense, this setting is also the counterpart to typical computations protected through trusted execution environments (TEEs). In comparison, SILENZIO does not rely on the additional hardware-based security assumptions bound to TEEs and are often shown to be vulnerable, especially through side-channel attacks, like e.g., [46], [47].

### B. Multi-Layer Perceptrons

Multi-Layer Perceptrons (MLPs) are a class of artificial neural network (NN) that consists of multiple layers of neurons, organized in a feedforward architecture. MLPs serve as universal function approximators and are widely used in various machine learning tasks, including classification and regression. In deep learning, MLPs serve as foundational building block for more complex architectures such as convolutional NNs and recurrent NNs.

An MLP is composed of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple neurons, which apply an affine transformation followed by a nonlinear activation function. Mathematically, for a given layer $l$, the output $\mathbf{h}^{(l)}$ is computed as: $\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)})$ where $\mathbf{W}^{(l)}$ represents the weight matrix, $\mathbf{b}^{(l)}$ is the bias vector, and $\sigma(\cdot)$ denotes a nonlinear activation function such as the ReLU function $\mathrm{ReLU} = \max(0, x)$.

Training an MLP involves adjusting the weights and biases to minimize a loss function, typically through backpropagation and gradient-based optimization methods like stochastic gradient descent (SGD).

### *C. Fully Homomorphic Encryption*

Fully Homomorphic Encryption is a form of encryption that allows computations to be performed directly on encrypted data without requiring decryption. This property is particularly useful for privacy-preserving computations, enabling secure outsourcing of computations to untrusted environments such as cloud-service-providers. Most practical FHE schemes are based on the Learning-With-Errors (LWE) or Ring-Learning-With-Errors (RLWE) problems, which add an error term to the secret to protect its confidentiality. FHE schemes support arbitrary computations on ciphertexts by enabling both addition and multiplication operations. Performing such arithmetic operations on ciphertexts accumulates the added noise until the ciphertexts are not decryptable anymore. To allow an arbitrary number of operations on a ciphertext, FHE schemes provide a so-called bootstrapping operation that reduces the noise and allows for more arithmetic operations to be performed. An FHE scheme typically consists of the following suite of algorithms:

- **Key Generation** $\mathrm{KeyGen}(\lambda) \rightarrow (pk, sk)$: Outputs a public evaluation and secret en-/decryption key based on the given security parameter $\lambda$.
- **Encryption** $\mathrm{Enc}(p, sk) \rightarrow c$: Transforms a plaintext input into a ciphertext, using the secret encryption key.
- **Evaluation** $\mathrm{Eval}_f(c, pk) \rightarrow c'$: Given the public evaluation key allows a function $f$ to be performed on ciphertexts, generating encrypted results equivalent to those obtained by performing the same function on the plaintexts.
- **Decryption** $\mathrm{Dec}(sk, c) \rightarrow m$: Converts the computed ciphertext back into plaintext using the secret key.

*1) TFHE & Concrete:* We implement SILENZIO using Zama's Concrete library, that provides a Python-based interface to construct arbitrary integer-based circuits, which then are compiled to efficient TFHE-protected programs. While the circuit itself is arbitrary, Concrete is limited to 8 bit integer and using RNS representations under the hood up to 16 bit integer computations. TFHE's bootstrapping is programmable, meaning it allows to perform arbitrary operations on the inputs during the bootstrapping process. Based on TFHE's programmable bootstrapping (PBS), Concrete provides next to the regular linear operations addition, subtraction, and multiplication, also non-linear operations modeled through table lookups. In Concrete, the security parameter $\lambda$ is fixed to 128 bit.

### *D. Number Systems*

To keep the operands of FHE operations in manageable value ranges, SILENZIO relies on the following number systems.

*1) Residue Number System (RNS):* The residue number system (RNS) is a non-weighted number system that represents integer using a set of relatively prime moduli. Given a set of moduli $\{m_1, m_2, \dots, m_k\}$, called the RNS base, an integer $x$ is uniquely represented by the tuple $(x_1, x_2, \dots, x_k)$, where: $x_i = x \bmod m_i, \text{ for } i \in [k]$. Given this base, the RNS has a cardinality of $\Pi_{i=1}^{k} m_i = M_k$, meaning it can represent $M_k$ distinct values uniquely.

Operations in an RNS can be performed independently on the individual residues without the need to handle intermediate carry values, making the RNS predestined for use with FHE schemes. More specifically, using an RNS allows breaking up large numbers into multiple smaller residues, accelerating FHE operations. RNS representations are widely used to speed up FHE-protected computations [48], [49].

*2) Mixed-Radix Number System (MRNS):* The Mixed-Radix Number System (MRNS) extends traditional positional numbering by allowing each digit to have a varying base. A number $x$ in an MRNS with the radix-base $\{r_1, r_2, \dots, r_k\}$ is represented as $x = x_1 + \sum_{i=2}^{k} x_i \prod_{j=1}^{i-1} r_j$ where the digits $x_i$ satisfy $0 \leq x_i < r_i$. Similar to the RNS system, the cardinality of the MRNS is $R_k = \Pi_{i=1}^{k} r_i$.

*3) RNS-to-MRNS Conversion:* A value represented in RNS can be converted into an associated MRNS [50]. An RNS and an MRNS are called associated if for the set of moduli $\{m_0, m_1, \dots, m_k\}$ that define the RNS and the set of radices $\{r_0, r_1, \dots, r_k\}$ that define the MRNS we have $\forall i \in [k] : m_i = r_i$. Algorithm 1 shows the conversion process of a number $x$ given in RNS representation to a number $y = x$ in an associated MRNS representation, as described by Szabó and Tanaka [50]. Table VII shows an example.

**Algorithm 1** RNS2MRNS

```
Input: $(x_1, \dots, x_k)$, $(m_1, \dots, m_k)$    ▷ x in RNS, RNS base
Output: $(y_1, \dots, y_k)$, $(r_1, \dots, r_k)$    ▷ x in MRNS, MRNS base
  $(y_1, \dots, y_k) \leftarrow (x_1, \dots, x_k)$
  for $i \leftarrow 2$ to $k$ do
      $(y_i, \dots, y_k) \leftarrow (y_i, \dots, y_k) - y_{i-1}$
      $(t_i, \dots, t_k) \leftarrow (m_{i-1}^{-1} \bmod m_i, \dots, m_{i-1}^{-1} \bmod m_k)$
      $(y_i, \dots, y_k) \leftarrow (y_i, \dots, y_k) \cdot (t_i, \dots, t_k)$
  end for
  $(r_1, \dots, r_k) \leftarrow (m_1, \dots, m_k)$    ▷ Associated MRNS base
```

## III. SILENZIO

We introduce our approach for non-interactive outsourcing of MLP training without any non-collusion assumptions.

### *A. Threat Model*

We assume a scenario where the client outsources the training of an MLP to an untrusted cloud-service-provider under a semi-honest, or honest-but-curious, adversary model. The provider strictly follows the protocol but may attempt to learn information from the data it processes. All training data and model parameters are encrypted using TFHE configured to achieve 128 bit security. For reusing encrypted training data and the successive updating of the weights during training, we rely on TFHE's security guarantees enabling the secure

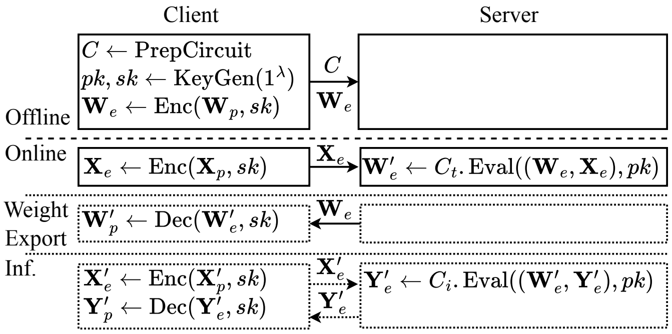
TABLE I
NOTATIONS

| Notation | Description |
|---|---|
| $\mathcal{U}_b$ | $\mathbb{Z} \cap [0, 2^b]$ |
| $\mathcal{I}_b$ | $\mathbb{Z} \cap [-2^{b-1}, 2^{b-1} - 1]$ |
| $\mathcal{J}_b$ | $\mathcal{U}_b$ values representing a number in RNS |
| $\mathcal{K}_b$ | $\mathcal{U}_b$ values representing a number in MRNS |
| $\mathbf{m}$ | RNS base: vector of modules |
| $w$ | Bitwidth of the RNS modules |
| $\alpha$ | Signed bitwidth of the model parameters |
| $\beta$ | Signed bitwidth of the inferece/training data |
| $\Gamma$, $\gamma$ | Signed / unsigned output bitwidth |
| $x$ | Cap value of $\mathrm{ReLU}_x$ |

reuse of encrypted inputs. For our implementation of SILENZIO, we leveraged Zama's Concrete Library with the default parameters, which means the implementation is secure in the IND-CPA security model. Concretely, the cloud customer, aka model owner, should not share the results of the outsourced computation (e.g., the trained model or inference results) in cleartext with third parties, as this could leak the secret key. The confidentiality of the system relies on the assumption that the client securely stores the secret key and never discloses it to any third party, thereby preventing the provider from decrypting any sensitive information. This model is designed solely to preserve confidentiality and does not extend to protect against active or integrity-based attacks, as it does not include mechanisms for detecting protocol deviations. Although our threat model is similar to that of FHESGD [13] and Glyph [14], which also operates under a semi-honest assumption, our approach significantly strengthens the security guarantee by employing a realistic 128 bit security level in contrast to FHESGD's and Glyph's 80 bit setting.

### B. Setup

As visualized in Figure 1, SILENZIO is set up in two stages, we call offline and online phase. In the input-independent *offline phase*, the client prepares the public evaluation and secret en-/decryption keys. Further, they prepare the TFHE-protected circuit, including the randomly initialized and encrypted weights, and transmit them to the remote cloud-service-provider. In the *online phase*, the client sends their encrypted dataset to the server, and the server performs the protected training. After the training is finished, the client can either use the encrypted weights on the server or request the server to send the trained weights back for decryption and local inference usage. For low-power devices like smart sensors in the IoT context, it's also thinkable that the client sends new training data points continuously to unburden local storage resources and keep the outsourced MLP model through continuous training up-to-date. Note that in SILENZIO and its evaluation, we concentrate on the training and do not consider the inference phase. To accelerate the inference after the training phase, it might be interesting to explore ways of combining the trained weights of SILENZIO with a more inference-focused approach like REDSec [51].



Client
Server
$C \leftarrow \mathrm{PrepCircuit}$
$pk, sk \leftarrow \mathrm{KeyGen}(1^\lambda)$
$\mathbf{W}_e \leftarrow \mathrm{Enc}(\mathbf{W}_p, sk)$
$C$
$\mathbf{W}_e$
Offline
Online
$\mathbf{X}_e \leftarrow \mathrm{Enc}(\mathbf{X}_p, sk)$
$\mathbf{X}_e$
$\mathbf{W}'_e \leftarrow C_t.\mathrm{Eval}((\mathbf{W}_e, \mathbf{X}_e), pk)$
Weight Export
$\mathbf{W}'_p \leftarrow \mathrm{Dec}(\mathbf{W}'_e, sk)$
$\mathbf{W}_e$
Inf.
$\mathbf{X}'_e \leftarrow \mathrm{Enc}(\mathbf{X}'_p, sk)$
$\mathbf{Y}'_p \leftarrow \mathrm{Dec}(\mathbf{Y}'_e, sk)$
$\mathbf{X}'_e$
$\mathbf{Y}'_e$
$\mathbf{Y}'_e \leftarrow C_i.\mathrm{Eval}((\mathbf{W}'_e, \mathbf{Y}'_e), pk)$

Fig. 1. SILENZIO's set up divided into an input-independent offline and an online phase. Here, we denote plaintexts and ciphertexts with a subscript $p$ respective $e$ and indicate the evaluation of a circuit through $C.\mathrm{Eval}()$.

TABLE II
RECENT NEURAL NETWORK FRAMEWORKS IN THE NON-INTERACTIVE OUTSOURCING SETTING SORTED BY TARGET WORKLOAD. SOME FRAMEWORK NAMES ARE MADE UP FOR EASIER REFERENCING. LFE IN FINE-TUNING MEANS THE CLIENT HAS TO PERFORM FEATURE EXTRACTION WITH A PUBLIC PRE-TRAINED MODEL LOCALLY AND THEN SENDS THESE FEATURES TO THE SERVER TO TRAIN A NEW MODEL. *ACCORDING TO THE AUTHORS, BLINDTUNER WILL BE OPEN SOURCED UPON ACCEPTANCE. HETAL COMMUNICATES THE MODEL OUTPUT OF A VALIDATION SET AFTER EACH EPOCH TO PROVIDE A CLIENT-DRIVEN EARLY-STOPPING FEATURE, BUT APART FROM THAT FEATURE, HETAL IS ALSO NON-INTERACTIVE.

| | Name | Scheme | Enc. Weights | Sec. Level | Open Source |
|---|---|---|---|---|---|
| **Inference** | CryptoNets [52] | YASHE | ○ | 80 | ● |
| | Faster CryptoNets [53] | FV-RNS | ○ | 128 | ○ |
| | FHE-DiNN [54] | TFHE | ○ | 80 | ● |
| | LoLa [55] | BFV | ○ | 128 | ● |
| | MPCNN [56] | CKKS | ○ | 128 | ○ |
| | ResFHE [57] | CKKS | ○ | 111.6 | ○ |
| | REDsec [51] | TFHE | ● | 128 | ● |
| | AutoFHE [58] | CKKS | ○ | 128 | ● |
| | DaCapo [59] | CKKS | ○ | 128 | ● |
| | NeuJeans [60] | CKKS | ○ | 128 | ○ |
| | Nexus [61] | CKKS | ● | 128 | ● |
| | LowMemInf [62] | CKKS | ○ | 128 | ● |
| | LOHEN [63] | CKKS, TFHE | ○ | 128 | ○ |
| | FHE-Neuron [64] | TFHE | ○ | 128 | ○ |
| **Fine-t.** | PrivGD [65] (LFE) | CKKS | ● | 80 | ○ |
| | HETAL [66] (LFE) | CKKS | ● | 128 | ● |
| | BlindTuner [67] (LFE) | CKKS | ● | 128 | * |
| | Glyph [14] | TFHE, BGV | ● | 80 | ○ |
| **Train.** | FHESGD [13] | BGV | ● | ∼ 80 | ○ |
| | Glyph [14] | TFHE, BGV | ● | 80 | ○ |
| | SILENZIO | TFHE | ● | 128 | ● |

● Support ○ No Support

### C. Encoding & Bitwidths

Through NITI, Wang *et al.* [16] demonstrated the feasibility of training NNs using 8 bit weights while down-scaling activations, gradients, and the errors also equal to or below 8 bits. Previous works like [68]–[75] showed the effectiveness of using integer weights below 8 bits for training, but their activations, errors, or gradients are much larger or quantized

TABLE III
RNS BASES USED IN SILENZIO.

| $k$ | RNS Base | Max. Bitwidth |
|---|---|---|
| 2 | 15, 14 | 7.72 |
| 3 | 13, 15, 14 | 11.41 |
| 4 | 11, 13, 15, 14 | 14.87 |
| 5 | 7, 11, 13, 15, 8 | 16.87 |
| 6 | 5, 7, 9, 11, 13, 8 | 18.45 |

from 32 bit floating-point values, making them unsuitable for our integer-only-based training approach. Following NITI, SILENZIO leverages $\alpha \leq 8$ bit signed integer weights for effective training. To make SILENZIO's whole end-to-end training pass low-bitwidth compatible, we additionally propose a new low-bitwidth gradient computation for the cross-entropy loss, further reducing the bitwidth requirements compared to NITI.

State-of-the-art FHE implementations, like the Concrete library, are most effective for bitwidths smaller or equal to 8 bits. To allow effective training while maintaining good runtime performance, SILENZIO only computes on tiny integer values for the entire computation, never exceeding 8 bits, and only using RNS representations where needed. To constrain the bitwidth of the used values below or equal to 8 bits and enable SILENZIO's optimized operations, we leverage the RNS bases shown in Table III for the evaluation. Note that SILENZIO also supports larger bitwidths as shown in Table VIII and explained in Section III-D2.

To summarize, we follow previous work from the hardware community [16] showing the effectiveness of training using low-bitwidth integer weights and errors and exploit it to boost the runtime performance of FHE-protected training. SILENZIO works with integer in the finite ring $\mathbb{Z}_{M_k}$. To represent signed integer, positive values are assigned to the lower half of $\mathbb{Z}_{M_k}$, and negative values to the upper half. The mapping $(0, 1, 2, -2, -1) \mapsto (0, 1, 2, 3, 4)$ illustrates the encoding.

### D. Training Scheme

In the following, we introduce each of SILENZIO's components in detail to finally outline the whole end-to-end training approach. For simplicity, we leverage NumPy's broadcasting and indexing rules [76] in the descriptions of the algorithms. Particularly, the $\mathrm{expandDims}(tensor, axis)$ function has the same syntax and semantics as in NumPy, changing the dimensions of a tensor without changing the data. Additionally, we leverage the FHE components described in Section II-C and Concrete's efficient implementations for modular reduction $\mathrm{mod}$, $\max$, $\mathrm{ReLU}$ and bit-extraction $\mathrm{extractBits}(input, indices)$ as black boxes. We denote our custom lookup tables for use with Concrete's PBS as $\mathrm{lookupComp}()$ and multivariate functions with two encrypted inputs as $\mathrm{multivariateLookupComp}()$. Table I introduces the used notations. Figure 2 shows the interplay of SILENZIO's components and the used number systems from a high-level perspective.

*1) $ReLU_x$:* To keep the non-linear activations of our models in manageable value ranges, we follow the approach of Krizhevsky *et al.* [77], cap the ReLU function and call it now

$$\mathrm{ReLU}_x(a) = \min(\max(a, 0), x) \text{ with } x \geq 0.$$

While Krizhevsky *et al.* fixed $x = 6$, we evaluated SILENZIO for a wide range of cap values $x \in [0, 127]$ and found $x = 14$ to generally be a suitable threshold for SILENZIO's low-bitwidth integer training approach.

*2) Low-bitwidth Matrix Multiplication:* As shown in Figure 2, the matrix multiplication $\mathrm{Matmul}^{\mathrm{RNS}}$ in SILENZIO gets up to 8 bit matrices on the forward- as well as the backward-pass as inputs. To keep all FHE-processed values $\leq$ 8 bit, we perform a conversion of both inputs into an RNS system (see also Algorithm 2). Since the used RNS bases consist of 4 bit moduli, all residues are $\leq$ 4 bit and we can compute the partial products in step 2 without the intermediate results overflowing 8 bit. Finally, the summation of the matrix multiplication is performed in the last step. We split the summation into modularly reduced sub-sums of maximal 15 summands to not overflow our limit of 8 bit intermediate results. Finally, all these sub-sums are summed up (modulo RNS base) to form the final result. Note that we can select the used RNS base individually per $\mathrm{Matmul}$ operation based on the required bitwidth. A computation including multiple $\mathrm{Matmul}$ operations of different input bitwidths and dimensions, like MLP training, greatly benefits in terms of computational efficiency from a heterogeneous set of RNS bases.

As part of SILENZIO, we additionally developed a high-resolution variant, $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$, which supports much larger bitwidths for the output of the matrix multiplication and may enable future work to train larger neural networks like, e.g., CNNs. $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$ leverages a new Karatsuba-inspired multiplication routine enabling the usage of 5 bit moduli in the RNS base and is fully compatible with all other components of SILENZIO. See Section B for a detailed description, including a numerical step-by-step example in Figure 7.

*3) Sign Determination in RNS:* To extract the sign information of a value represented in an RNS, we start with a simple and widely used approach. We first convert the number to the associated MRNS using our vectorized implementation of the RNS2MRNS conversion shown in Algorithm 1. Subsequently, we exploit that our encoding divides the finite ring $\mathbb{Z}_{M_k}$ of all possible values into the lower half for positive and the upper half for negative values. Now, we can extract the sign information using a single table lookup on the most significant radix position: $x_k \geq r_k/2$. To achieve full correctness, the sign extraction requires the most significant radix to be even. The most significant modulus of our RNS bases is always chosen to be even (see Table III) and as the RNS2MRNS conversion results in an MRNS representation of an associated MRNS base, we always meet this requirement and achieve correct sign extraction.

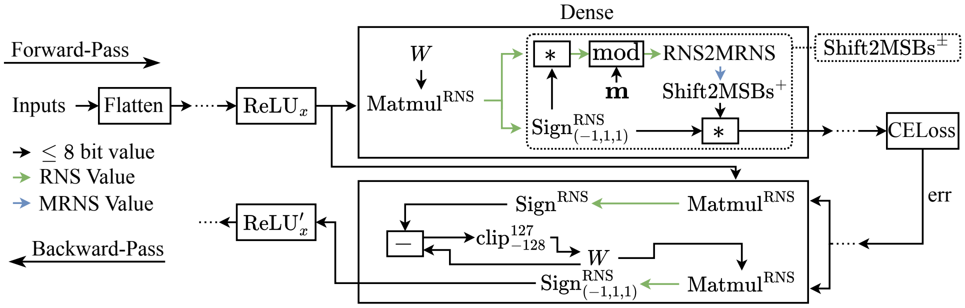

Dense
Forward-Pass
Inputs
Flatten
$\mathrm{ReLU}_x$
$W$
$\mathrm{Matmul}^{\mathrm{RNS}}$
$*$
mod
RNS2MRNS
$\mathbf{m}$
$\mathrm{Shift2MSBs}^{+}$
$\mathrm{Sign}^{\mathrm{RNS}}_{(-1,1,1)}$
$*$
$\mathrm{Shift2MSBs}^{\pm}$
CELoss
err
≤ 8 bit value
RNS Value
MRNS Value
$\mathrm{ReLU}'_x$
Backward-Pass
$\mathrm{Sign}^{\mathrm{RNS}}$
$\mathrm{Matmul}^{\mathrm{RNS}}$
$-$
$\mathrm{clip}^{127}_{-128}$
$W$
$\mathrm{Sign}^{\mathrm{RNS}}_{(-1,1,1)}$
$\mathrm{Matmul}^{\mathrm{RNS}}$

Fig. 2. High-level overview of the back- and forward-pass in SILENZIO.

**Algorithm 2** $\mathrm{Matmul}^{\mathrm{RNS}}$

```
Input: $\mathbf{X} \in \mathcal{I}_8^{a \times b}$, $\mathbf{W} \in \mathcal{I}_8^{b \times c}$, $\mathbf{m} \in \mathcal{U}_4^k$    ▷ Inputs, RNS base
Output: $\mathbf{Y} \in \mathcal{J}_4^{k \times a \times b}$    ▷ Output in RNS representation
  ▷ 1. Step: Convert inputs to RNS representation
  $\mathbf{m}' \leftarrow \mathrm{expandDims}(m, [1, 2])$
  $\mathbf{X}' \leftarrow \mathbf{X} \bmod \mathbf{m}'$    ▷ Shape: $k \times a \times b$
  $\mathbf{W}' \leftarrow \mathbf{W} \bmod \mathbf{m}'$    ▷ $k \times b \times c$
  ▷ 2. Step: Compute partial products
  $\mathbf{X}' \leftarrow \mathrm{expandDims}(\mathbf{X}', 3)$    ▷ $k \times a \times b \times 1$
  $\mathbf{W}' \leftarrow \mathrm{expandDims}(\mathbf{W}', 1)$    ▷ $k \times 1 \times b \times c$
  $\mathbf{Y}^* \leftarrow \mathbf{X}'\mathbf{W}' \bmod \mathrm{expandDims}(m, [1, 2, 3])$    ▷ $k \times a \times b \times c$
  ▷ 3. Step: Block-wise summation
  $n \leftarrow \min(15, b)$
  $\mathbf{Y} \leftarrow \mathrm{sum}(\mathbf{Y}^*[:, :, : n, :], axis = 2) \bmod \mathbf{m}'$
  for $i \leftarrow n,\ i \leq b,\ i \leftarrow i + n$ do
      $j \leftarrow \min(i + n, b)$
      $\mathbf{Y}' \leftarrow \mathrm{sum}(\mathbf{Y}[:, :, i : j, :], axis = 2)$
      $\mathbf{Y} \leftarrow \mathbf{Y} + \mathbf{Y}' \bmod \mathbf{m}'$    ▷ $k \times a \times c$
  end for
```

Additionally, SILENZIO can check for zero-equality. Therefore, we add the values of the MRNS representation along all radices and compare to zero. As our RNS bases, and so our MRNS bases never contain more than six 4 (or 5) bit moduli, their sum never exceeds 8 bit. Note that we could similarly construct the zero-equality check based on the RNS representation, but as we cannot jump (in the algorithm) based on encrypted values, there is no benefit to doing so. Leveraging the correct sign extraction and the zero-equality check, SILENZIO supports sign gadgets with the following semantics:

$$\mathrm{Sign}^{\mathrm{RNS}}_{(n,z,p)}(x) := \begin{cases} n, & \text{for } x < 0 \\ z, & \text{for } x = 0 \\ p, & \text{for } x > 0. \end{cases}$$

*4) SHIFT2MSBs$^{+}$:* One of SILENZIO's key features is a new block-scaling gadget that allows us to approximately shift an input matrix given in RNS representation to its user-specified $\gamma$ most significant bits. Therefore, the gadget approximates the maximum bitwidth present in the input and performs a shift operation based on this maximum. The gadget is central to keeping all FHE-processed values ≤ 8 bit while performing a high-resolution forward-pass and enabling a fruitful backward-pass that allows not only protected inference, but training resulting in accurate MLP models.

We start by describing SHIFT2MSBs$^{+}$ which allows shifting positive RNS represented matrices, and subsequently introduce SHIFT2MSBs$^{\pm}$ which also considers negative values. Algorithm 3 shows SHIFT2MSBs$^{+}$ and Table IV illustrates the algorithm exemplary step-by-step (without the RNS-to-MRNS conversion step). The algorithm gets a positive valued matrix in RNS representation $\mathbf{X}$ and the user-specified unsigned output bitwidth $\gamma$ as inputs. Note the unsigned output bitwidth $\gamma$ is just one bit less than the signed output bitwidth $\Gamma$. In the first step, the input matrix is converted from an RNS to the associated MRNS representation. From now on, the main idea of the gadget is to view the MRNS values as if they were chunked binary numbers, where each chunk is given by a radix-digit (see the second column in the example Table IV).

Based on the MRNS representation, we approximately extract the position of the highest set bit, $maxBit$, in step 2. Step 3 computes the amount of shift needed per radix-digit $digitShift$ to achieve an output bitwidth $\gamma$. First we check, whether any shift is required by comparing $maxBit$ to the radix bitwidth $w$, as we only want to shift down but never up. Shifting up would not provide any additional information to the training process. Then we compute the shift required per radix digit $digitShift$ based on $maxBit$, the required output bitwidth $\gamma$ and the maximum bitwidth per radix-digit $maxBW$. Finally, we compute the amount of required left and right shift based on $digitShift$.

In the fourth step, we just apply the left and right shifts per radix-digit on the MRNS representation. To get the final result, we sum over all radix-digits in step 5. Notice that step 4 and step 5 inherently convert the matrix from an MRNS to the classical decimal representation. We approximately compute the effectively performed shift amount $shift$ in the last step of the algorithm.

As the radices in our MRNS representations are not all powers of two, the approximation error of the gadget stems from the underlying idea of viewing the radix-digits as if they would make up chunks of a binary number. The error has two

**Algorithm 3** Shift2MSBs$^+$

```
Input: $\mathbf{X} \in \mathcal{J}_4^{k \times a \times b}$, $\mathbf{m} \in \mathcal{U}_4^k$, $w$, $\gamma$   ▷ Inputs, RNS base, RNS modul bitwidth, Unsigned output bitwidth
Output: $\mathbf{Y} \in \mathcal{U}_\gamma^{a \times b}$, $shift$   ▷ Output, Shiftamount
  ▷ 1. Step: RNS-to-MRNS conversion
  $\mathbf{X}', \_ \leftarrow \text{RNS2MRNS}(\mathbf{X}, \mathbf{m})$
  ▷ 2. Step: Extract the position of the highest set bit (1-indexed)
  $maxBit \leftarrow 0$
  for $i \leftarrow 0, i < k, i \leftarrow i + 1$ do
      $\mathbf{u} \leftarrow \text{lookupComp}(\lceil \log_2(\mathbf{X}'[i] + 1) \rceil + i \cdot w)$
      $maxBit \leftarrow \max(\max(\mathbf{u}), maxBit)$
  end for
  ▷ 3. Step: Compute amount of digitShift
  $anyShift \leftarrow maxBit > w$   ▷ Only shift down, never up
  for $i \leftarrow 0, i < k, i \leftarrow i + 1$ do   ▷ Can be pre-computed
      $maxBW[i] \leftarrow w \cdot i$
  end for
  $digitShift \leftarrow (\gamma - (maxBit - maxBW)) \cdot anyShift$
  $lshift \leftarrow \min(\max(digitShift, 0), \gamma - 1)$
  $rshift \leftarrow \text{ReLU}(-digitShift)$
  ▷ 4. Step: Perform Shift
  $\mathbf{Y}^* \leftarrow \mathbf{X}' \ll \text{reshape}(lshift, [k, 1, 1])$
  $\mathbf{Y}^* \leftarrow \mathbf{Y}^* \gg \text{reshape}(rshift, [k, 1, 1])$
  ▷ 5. Step: Reduce with final summation
  $\mathbf{Y} \leftarrow \text{sum}(\mathbf{Y}^*, axis = 0)$
  ▷ 6. Step: Compute amount of shift
  $shift \leftarrow maxBit - \gamma$
```

TABLE IV
EXAMPLE COMPUTATION OF SHIFT2MSBS$^+$ WITH AN INPUT OF SIZE 3 AND A BATCH OF A SINGLE SAMPLE. UNSIGNED OUTPUT BITWIDTH $\gamma = 5$ AND RNS MODUL BITWIDTH $w = 4$. SHIFT RESULTS ARE UNDERLINED AND FOR BREVITY, THE RNS-TO-MRNS CONVERSION IS OMITTED.

| MRNS Base | $r_2 = 14$<br>1110 | $r_1 = 15$<br>1111 | $r_0 = 13$<br>1101 |
|---|---|---|---|
| $\mathbf{X}'[:, 0, 0]$<br>$\mathbf{X}'[:, 0, 1]$<br>$\mathbf{X}'[:, 0, 2]$ | $00\underline{11}$<br>$00\underline{01}$<br>$00\underline{00}$ | $\underline{001}0$<br>$\underline{110}0$<br>$\underline{000}1$ | 0000<br>0010<br>0110 |
| $anyShift$ | $(maxBit > w) = (10 > 4) \implies$ True | | |
| $digitShift$<br>$lShift$<br>$rshift$ | $5 - (10 - 8)$<br>3<br>0 | $5 - (10 - 4)$<br>0<br>1 | $5 - (10 - 0)$<br>0<br>5 |
| $shifted$ | $\mathbf{X}'[:, 0, 0]$<br>$\underline{11}000$<br>$00\underline{001}$<br>00000 | $\mathbf{X}'[:, 0, 1]$<br>$\underline{01}000$<br>$00\underline{110}$<br>00000 | $\mathbf{X}'[:, 0, 1]$<br>$\underline{00}000$<br>$00\underline{000}$<br>00000 |
| $\mathbf{Y}$ | $\underline{11001}$ | $\underline{01110}$ | $\underline{00000}$ |
| $shift$ | $maxBit - \gamma = 10 - 5 = 5$ | | |

**Algorithm 4** Shift2MSBs$^\pm$

```
Input: $\mathbf{X} \in \mathcal{J}_4^{k \times a \times b}$, $\mathbf{m} \in \mathcal{U}_4^k$, $w$, $\Gamma$   ▷ Inputs, RNS base, RNS modul bitwidth, Signed output bitwidth
Output: $\mathbf{Y} \in \mathcal{I}_\Gamma^{a \times b}$, $shift$   ▷ Output, Shiftamount
  $s \leftarrow \text{Sign}^{\text{RNS}}_{(-1,1,1)}(\mathbf{X})$   ▷ Extract sign
  $\mathbf{X}^+ \leftarrow s \cdot \mathbf{X} \bmod \text{reshape}(\mathbf{m}, [k, n, n])$   ▷ Extract abs. values
  $\mathbf{Y}^+, shift \leftarrow \text{Shift2MSBs}^+(\mathbf{X}^+, \mathbf{m}, w, \Gamma - 1)$
  $\mathbf{Y} \leftarrow s \cdot \mathbf{Y}^+$   ▷ Add sign back to result
```

sources: first, the approximative determination of the highest set bit, and second, the per radix-digit bit-shifts together with the final digit summation, which do not consider the semantics of the MRNS representation. We analyze the approximation error as part of our evaluation in the practical application of MLP training.

*5) SHIFT2MSBs$^\pm$:* To allow the shifting of negative numbers in RNS representation, we introduce SHIFT2MSBs$^\pm$ shown in Algorithm 4. The main idea is that shifting in the negative domain is congruent to shifting in the positive domain and that our encoding allows for efficient computation of absolute values given a number in RNS representation.

The algorithm gets an input matrix $\mathbf{X}$ in RNS representation and the signed output bitwidth $\Gamma$ as inputs. First, we extract the sign information $s$ of the input using the approach introduced in Section III-D3. Using the sign information, we compute the absolute values $\mathbf{X}^+$ of the input in RNS representation. Afterward, we use our SHIFT2MSBs$^+$ algorithm to shift the absolute values to the $\Gamma - 1$ most significant bits. In the last step, we reapply the sign information, extracted in the beginning, to the shifted absolute values $\mathbf{Y}^+$ resulting in the final output $\mathbf{Y}$.

*6) Integer Cross-Entropy Loss Derivative:* Given the logits $a_i$ at the output of the final layer of an MLP, the corresponding softmax distribution is defined as $\hat{y}_i = e^{a_i} / \sum_i e^{a_i}$. We can now compute the error based on the cross-entropy loss, which is defined as $L = -\sum_i y_i \ln(\hat{y}_i)$ with $\mathbf{y}$ being the target probabilities given through the labels in the training data. The error is computed as the partial derivatives of the loss with respect to the MLP outputs

$$\partial L / \partial a_i = \hat{y}_i - y_i.$$

As shown in Algorithm 5 SILENZIO approximates the error computation, depending solely on integer computations. The algorithm expects the logits on the output of the last layer in the MLP $\hat{\mathbf{Y}}$, the corresponding one-hot encoded labels of the training dataset $\mathbf{Y}$ and an approximation level $\kappa$ for the exponential function. First, we approximate the exponential function using $\lceil \log_2(2^\kappa + 1) \rceil$ bit integer through a table lookup. Figure 3 shows a plot of the approximation for different approximation levels $\kappa$. As we sum these approximations in the next step and still hold the requirement of not exceeding 8 bit intermediate values, the algorithm is limited to MLPs with up to $\log_2(o \cdot 2^\kappa + 1) \leq 8 \implies o \leq 255/2^\kappa$ output neurons. To compute the error, we leverage a multivariate table lookup with two encrypted inputs and subtract the target "probabilities" given through the training labels. Finally, for training MLPs, SILENZIO just extracts the sign of the error and leverages it for the backpropagation algorithm. Compared

**Algorithm 5** IntCELossDeriv

```
Input: $\hat{\mathbf{Y}} \in \mathcal{I}_{\Gamma}^{a\times b}$, $\mathbf{Y} \in \mathcal{U}_{1}^{a\times b}$, with $b \leq 16$, $\kappa$   ▷ Logits, One-hot encoded label, exp. approx. level
Output: $\mathbf{E} \in \mathcal{I}_{\lceil \log_2(b\cdot 2^{\kappa}+1)\rceil}^{a\times b}$   ▷ Error
  ▷ round(x) means rounding x to the next integer. If x is exactly between two integer we round to the nearest even integer.
  $\mathbf{E} \leftarrow \mathrm{ReLU}(\hat{\mathbf{Y}})$
  $\mathbf{E} \leftarrow \mathrm{lookupComp}(\mathrm{round}((e^{\hat{\mathbf{Y}}})/e^{2^{\gamma}-1} \cdot 2^{\kappa}))$
  $\mathbf{S} \leftarrow \mathrm{reshape}(\mathrm{sum}(\mathbf{E}, axis = 1), [a, 1])$
  $\mathbf{E} \leftarrow \mathrm{multivariateLookupComp}(\mathrm{round}(\mathbf{E} \cdot 2^{\kappa} + 1/\mathbf{S} + 1))$
  $\mathbf{E} \leftarrow \mathbf{E} - \mathbf{Y} \cdot \mathrm{reshape}(\mathrm{sum}(\mathbf{E}, axis = 1), [a, 1])$
  $\mathbf{E} \leftarrow \mathrm{lookupComp}((\mathbf{E} > 0)\text{-}(\mathbf{E} < 0))$   ▷ Extract sign
```

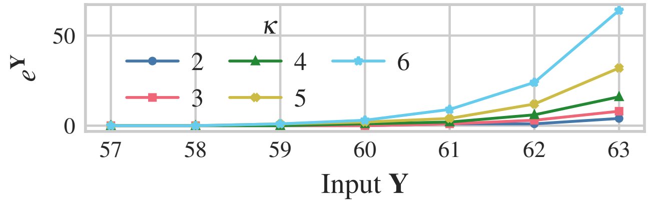


Fig. 3. Input-output behavior of our approximated exponential function $\exp(\mathbf{Y}) = \mathrm{round}((e^{\mathbf{Y}})/e^{2^{6}-1} \cdot 2^{\kappa})$ on signed 7 bit inputs for various approximation level $\kappa$.

to the derivative computation in NITI, SILENZIO depends on much smaller integer, never exceeding 8 bit.

*7) End-to-End Training:* Algorithm 6 shows SILENZIO's end-to-end training approach for a single data batch and for brevity without activation functions. SILENZIO leverages stochastic gradient descent based optimization for effective training. Training over multiple data batches and epochs can be performed through repeated execution of the training pass while continuously updating the same encrypted weights $\mathbf{W}$. Before every matrix multiplication, we perform an unencrypted table lookup using getRNSBase() on Table III to set the used RNS base for the required operand resolution, determined by the input dimensions and bitwidths.

On the forward-pass, we leverage Shift2MSBs$^{\pm}$ to reduce the bitwidth of the matrix multiplication results back to $\Gamma$ bits while simultaneously converting it from the RNS to the decimal number system. While Shift2MSBs$^{\pm}$ also provides the amount of shift performed, SILENZIO does not use it for the training. As demonstrated by Wang *et al.* [16] in an unprotected setting, the amount of shift could be used in future work to construct schemes to train ML models which require the amount of shift, also known as scaling exponent in fixed-point arithmetic, for effective training.

We instantiate the back-propagation of the error using our approximated cross-entropy loss derivation. During MLP-Training SILENZIO just leverages the sign of the error for backpropagation. Similar to previous work [16], [78] we perform the weight update based on the sign of the weight gradients. We leverage Sign$^{\mathrm{RNS}}$ and extract the sign information directly from the outputs of the Matmul$^{\mathrm{RNS}}$ computations, omitting additional Shift2MSBs$^{\pm}$ operations for downshifting and number-system conversion. Finally, to limit the weights to 8 bit during the weight update and prevent intermediate results greater than 8 bit, we leverage a multivariate table lookup. If required, a bias value could be introduced through the bias-trick by appending a constant 1-entry to the input and adding a bias column to the weight matrix.

**Algorithm 6** End-to-End Training (Single Batch)

```
Input: $\mathbf{A}_0 \in \mathcal{I}_{\beta}^{a_0\times b_0}$, $\mathbf{Y} \in \mathcal{U}_{1}^{a\times b}$, $\alpha$, $\beta$, $\Gamma$, $x$   ▷ Data batch, One-hot enc. labels, model bitwidth, input bitwidth, signed output bitwidth, ReLU$_x$ cap value
Output: $\mathbf{W}_l \in \mathcal{I}_{\alpha}^{c_l\times b_l}$ for all $l \in [1, L]$   ▷ MLP weights
  ▷ We train an MLP of L weight layers. For brevity, we omit the activation functions.
  ▷ Forward-Pass
  for $l \leftarrow 1, l \leq L, l \leftarrow l + 1$ do
      if $l = 1$ then
          $\mathbf{m} \leftarrow \mathrm{getRNSBase}(\log_2(b_{l-1} \cdot 2^{\beta} \cdot 2^{\alpha} + 1))$
      else
          $\mathbf{m} \leftarrow \mathrm{getRNSBase}(\log_2(b_{l-1} \cdot x \cdot 2^{\alpha} + 1))$
      end if
      $\mathbf{A}_l \leftarrow \mathrm{Matmul}^{\mathrm{RNS}}(\mathbf{A}_{l-1}, \mathbf{W}_l^{\top}, \mathbf{m})$   ▷ Layer l activation
      $\mathbf{A}_l \leftarrow \mathrm{Shift2MSBs}^{\pm}(\mathbf{A}_l, \mathbf{m}, 4, \Gamma)$   ▷ 4 bit RNS moduls
  end for
  ▷ Backward-Pass
  $\mathbf{E} \leftarrow \mathrm{IntCELossDeriv}(\mathbf{A}_l, \mathbf{Y})$
  for $l \leftarrow L, l \geq 1, l \leftarrow l - 1$ do
      if $l > 1$ then
          $\mathbf{m} \leftarrow \mathrm{getRNSBase}(\log_2(a_{l-1} \cdot 2 \cdot x + 1))$
          $\mathbf{G} \leftarrow \mathrm{Matmul}^{\mathrm{RNS}}(\mathbf{E}^{\top}, \mathbf{A}_{l-1}, \mathbf{m})$
          $\mathbf{m} \leftarrow \mathrm{getRNSBase}(\log_2(c_l \cdot 2^{\alpha} + 1))$
          $\mathbf{E} \leftarrow \mathrm{Matmul}^{\mathrm{RNS}}(\mathbf{E}, \mathbf{W}_l, \mathbf{m})$
          $\mathbf{E} \leftarrow \mathrm{Sign}_{(-1,0,1)}^{\mathrm{RNS}}(\mathbf{E})$   ▷ Sign for error propagation
      else
          $\mathbf{m} \leftarrow \mathrm{getRNSBase}(\log_2(a_{l-1} \cdot 2^{\beta} + 1))$
          $\mathbf{G} \leftarrow \mathrm{Matmul}^{\mathrm{RNS}}(\mathbf{E}^{\top}, \mathbf{A}_{l-1}, \mathbf{m})$
      end if
      $\mathbf{G} \leftarrow \mathrm{Sign}_{(-1,0,1)}^{\mathrm{RNS}}(\mathbf{G})$   ▷ Sign for weight update
      $\mathbf{W} \leftarrow \mathrm{multivariateLookupComp}(\mathrm{clip}_{-128}^{127}(\mathbf{W} - \mathbf{G}))$
  end for
```

## IV. IMPLEMENTATION

We describe our implementation, used to evaluate the individual components and SILENZIO's end-to-end performance.

### A. Setup

We used a server running Ubuntu 24.04 equipped with 2x AMD EPYC 9534 and the following software: Python 3.10.13, Concrete-Python 2.11.0, Jax 0.6.2, PyTorch 2.8.0 and scikit-learn 1.7.1. We leverage Concrete-Python to compile SILENZIO's components into an efficient TFHE program. To speed up the development and evaluation, we also implemented a fast clear-text clone based on Jax-accelerated NumPy that performs the same approximations as the FHE-based components. To compare the prediction-performance of the MLPs trained with Silenzio, we used standard PyTorch to construct models of the same architecture as a baseline.

### *B. Circuit Construction*

Congruent to classical NN frameworks like TensorFlow and PyTorch and, as depicted in Figure 2, Silenzio models MLPs in components of fully connected and activation layers. In the standard workflow of Concrete, one provides the computation circuit to be compiled to an TFHE program as a Python function. Concrete supports a wide range of standard Python functions and a solid subset of the NumPy functionality[1]. We developed the components described in Section III-D as individual Python functions and composed them to the forward- and backward-passes of the individual layers.

### *C. Tracing & Compilation*

As part of the compilation process, Concrete traces the circuit to determine the value ranges of all intermediate results. Silenzio pre-computes the compilation of the circuit during the input-independent offline phase using randomly generated input- and label-tensors.

Compiling large and complex circuits as required to train MLPs presents a challenge, as compilation times blow up because of excessive circuit-wide optimization. Compiling the whole for- and backward-pass into a single circuit is therefore not feasible. To circumvent the optimization overhead, we compiled the for- and backward-passes as well as weight updates of each layer and the error computation regarding the loss into separate circuits and composed them into a single training pass using Concrete's modules feature. Silenzio benefits from compiler optimizations inside each of these components, while keeping overall compile times manageable. To train an MLP model for multiple epochs and data batches, the server evaluates the training pass repeatedly while always using the updated weights of the previous iteration (also see Section III-A). We build a simple Keras-inspired interface to stack MLP-layers in Python by monkey-patching Concrete and enabling dynamic input-shapes for our layers.

## V. Evaluation

Similar to previous work, we evaluate Silenzio only regarding the online runtime performance and do not evaluate the communication costs of the inputs, as they are negligible in the context of the whole runtime overhead induced by FHE. To prevent stalling, in practice, one can start the training process after the first encrypted input batch arrives at the server, interlacing communication and computation.

In the following, we evaluate the performance of Silenzio in terms of online runtime for eight real-world MLP training tasks and compare the prediction performance of the resulting models to standard PyTorch training using 32 bit floating-point computations.

TABLE V
Summary of datasets: samples #S, features #F, classes #C, input bitwidth $\beta$. Summary of MLPs: bitwidth of model parameters $\alpha$, epochs #E, number of trainable parameters #P and architectures.

| Dataset | #S | #F | #C | $\beta$ | $\alpha$ | #E | #P | MLP |
|---|---|---|---|---|---|---|---|---|
| B. Cancer [81] | 569 | 30 | 2 | 4 | 5 | 25 | 1080 | 28-8-2 |
| T. Cancer [82] | 383 | 16 | 2 | 2 | 2 | 1 | 840 | 20-8 |
| Diabetes [83] | 768 | 8 | 2 | 4 | 5 | 25 | 144 | 8-8-2 |
| Wine [84] | 178 | 13 | 3 | 2 | 6 | 25 | 297 | 13-8-3 |
| V. Column [85] | 310 | 6 | 3 | 5 | 7 | 100 | 164 | 10-8-3 |
| Parkinsons [86] | 197 | 22 | 2 | 2 | 6 | 50 | 912 | 32-8-2 |
| H. Disease [87] | 303 | 13 | 2 | 2 | 5 | 50 | 289 | 13-8-2 |
| H. Failure [88] | 299 | 12 | 2 | 2 | 6 | 50 | 256 | 12-8-2 |

### *A. Datasets, Models & Hyperparameters*

We used the following datasets and MLP models for the evaluation. To validate the model performance, we used a 30% test split for the thyroid cancer dataset and a 20% test split for all other datasets. We leveraged uniform weight initialization, a signed output bitwidth $\Gamma = 7$, $\mathrm{ReLU}_x$ with $x = 14$ and a loss approximation level $\kappa = 4$ for all models trained using Silenzio. For the models trained with PyTorch, we used the default uniform weight initialization and classic ReLU activations. As we did not achieve the expected classification performance when training the PyTorch models using SGD [79], we use the Adam optimizer [80] with a learning rate of $lr = 0.001$. Similar to NITI [16], we train all models without bias values. For simplicity, we used a batch size of eight samples for all models. To represent categorical features, we used a one-hot encoding. Additionally, we performed some standard preprocessing on all datasets, including removing the mean of some features, replacing zero values with the mean, and transforming skewed value distributions with a quantile transformer[2]. To simplify the circuit construction and reduce compile times, we drop the last data batch of each epoch if the number of its samples is smaller than the specified batch size. Table V shows a summary of all used datasets and MLP configurations.

### *B. End-to-End Runtime*

To evaluate the online runtime of Silenzio, we follow the approach of previous work, e.g., like [13] and measure the runtime of our FHE-protected implementation of Silenzio for a single data batch and subsequently project it to the whole runtime, meaning the number of batches required to achieve the best respective PyTorch-level test accuracy. To ensure the correctness of the FHE-protected computations, we compared the resulting activations, errors, weight gradients, and the final weights of all layers with our NumPy-based implementation.

[1] During our development, not all functions worked as expected, but choosing an equivalent alternative, modeling the functionality directly as a lookup table or subdividing complex broadcast operations into smaller sub-operations often solved compilation problems.

[2] To ensure reproducibility, the exact preprocessing procedure is part of our open-source artifact.

As neither FHESGD [13] nor Glyph [14] provides an openly available implementation and both schemes only achieve 80 bit of security, thus being insecure, we refrain from a comparison. Using FHE parameters to achieve 80 bit security in the implementation of SILENZIO to "enable" a comparison is not useful as it would not be transferable to a secure setting[3]. Table VI reports the runtime results for all eight MLPs. As expected, the overhead induced by the FHE-protection with 128 bit security is substantial, but we argue it is still manageable and, in many scenarios, practical. While being orders of magnitude slower than clear text training, SILENZIO achieves security guarantees compelling for the outsourcing of training in many use cases handling highly sensitive data, like in the medical domain. In these settings, SILENZIO can enable the non-interactive outsourcing of training for the first time. Also, compared to TEE-based outsourced training, SILENZIO introduces considerable overhead. But again, we argue SILENZIO achieves much stronger security guarantees by never decrypting on the server-side and keeping the decryption key only stored on the client side, not exposing it to potential server-side side-channel attacks. Thus, SILENZIO can enable non-interactive outsourcing of training in scenarios, where TEE-based security guarantees are not suitable.

### C. Layer-wise Runtime Distribution

Figure 4 shows the runtime distribution over the individual circuits composing SILENZIO's training pass for all eight evaluated MLP models. The first thing to notice is that the large majority of the runtime is spent on the forward- and backward passes of the dense layers, while the activation functions, loss computations, and weight updates only insignificantly contribute to the end-to-end runtime in comparison. Secondly, it is noticeable that the forward-pass requires more than 60% and often close to 80% of the end-to-end runtime, although the backward-pass has to perform two matrix multiplications for the error and the weight gradient computations. We attribute the efficiency of the backward-pass to its much smaller resolution requirements and SILENZIO's ability to adaptively leverage much smaller RNS representations and efficient RNS-based sign extraction gadget $\mathrm{Sign}_{(-1,0,1)}$. Based on these results, future research should focus on further accelerating the linear layers to reduce the computational overhead of FHE-protected non-interactive outsourced training over its clear-text pendant.

### D. Classification Accuracy

To understand SILENZIO's ability to train accurate MLP models, we trained all eight MLPs using our fast clear-text NumPy-Implementation of SILENZIO and PyTorch. We report the best test accuracies achieved during the training in Table VI. SILENZIO achieves the same or better test accuracies for all datasets compared to the PyTorch-based 32-bit floating point training.

Figure 5 compares the development of the test accuracy during the training with SILENZIO to the training using PyTorch. Interestingly, SILENZIO's training converges faster than PyTorch in all cases. The accuracy developments of all eight models clearly demonstrate the effective learning ability of our low-bitwidth integer based training approach. To recap, SILENZIO achieves the same or better classification accuracy compared to PyTorch in all evaluated classification tasks, being a secure and accurate drop-in replacement for unprotected training while creating a solid foundation for future research aiming to further improve the effectiveness of integer-based non-interactive outsourced training.

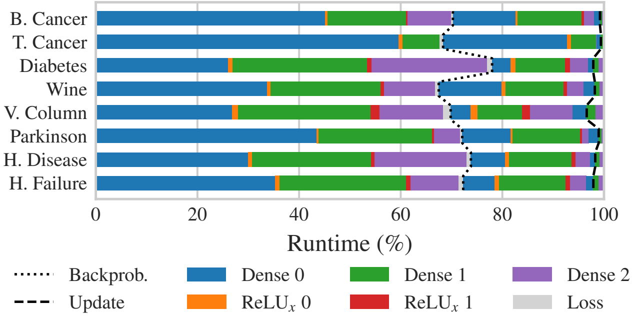


Fig. 4. Runtime distribution over the individual circuits composed to form SILENZIO's end-to-end training pass for the eight MLP models of our evaluation. The vertical lines indicate at which point the backpropagation and the weight update phases begin.

TABLE VI
SILENZIO'S RUNTIME PERFORMANCE AND TRAINING PERFORMANCE IN COMPARISON TO STANDARD PYTORCH. WE REPORT THE PROJECTED RUNTIME REQUIRED TO ACHIEVE THE BEST (LEFT) AND THE PYTORCH-LEVEL (RIGHT) TEST ACCURACY.

| Model | Test Acc.<br>PyTorch (fp32) | Test Acc.<br>SILENZIO | Time<br>(s/batch) | Projected<br>time (h) |
|---|---|---|---|---|
| B. Cancer | 96.5% | **98.3%** | 1027 | 19.4 / 17.7 |
| T. Cancer | **100.0%** | **100.0%** | 451 | 0.1 / 0.1 |
| Diabetes | 81.2% | **85.1%** | 295 | 212.6 / 31.7 |
| Wine | 94.4% | **100.0%** | 419 | 3.1 / 2.0 |
| V. Column | 74.2% | **80.7%** | 450 | 187.8/ 25.3 |
| Parkinsons | **89.7%** | **89.7%** | 924 | 43.6 / 43.6 |
| H. Disease | 80.3% | **86.9%** | 387 | 80.1 / 8.9 |
| H. Failure | 81.7% | **90.0%** | 361 | 40.1 / 11.7 |

### E. Approximation Impact

We quantify the approximation error of Shift2MSBs$^{\pm}$ by tracking its absolute difference to an exact block-scaling implementation during training. Figure 6 shows the error distribution of the dense layer activations during the training. While the error deviates significantly between layers and datasets, the mean of the error stays below ten for nearly all layers and MLPs except for the last layer of the heart disease MLP, which underlies, in general, large errors. On average the absolute error is 4.7, which, considering that we trained all models with a signed output bitwidth $\Gamma = 7$, equates to a relative error of 3.7%. The extreme error cases are substantial, with up to an absolute value of 28, or a relative error of 21.9%.

[3] Concrete only supports secure execution with 128-bit security

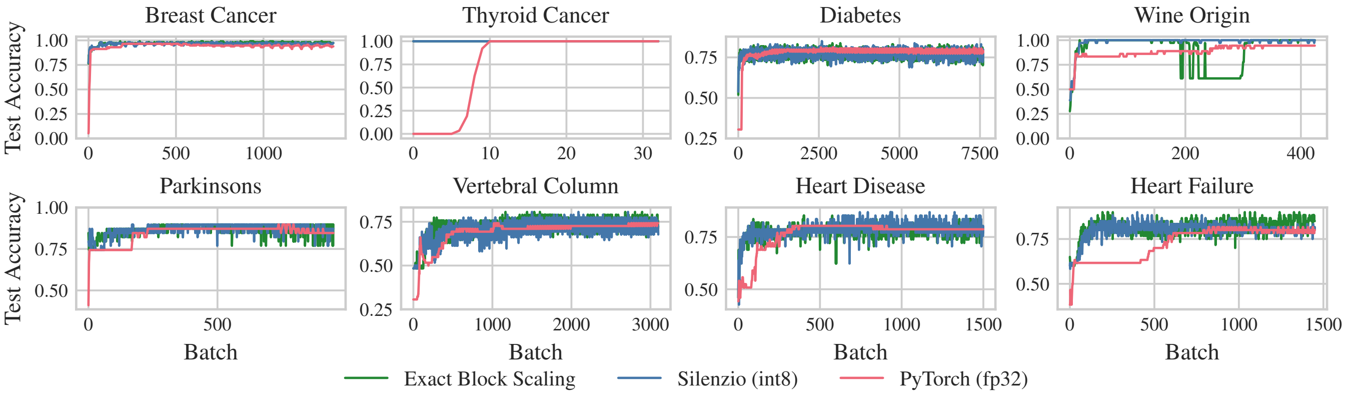


Fig. 5. Test accuracy of MLPs trained using SILENZIO with Shift2MSBs$^{\pm}$ or exact block-scaling compared to PyTorch (fp32).

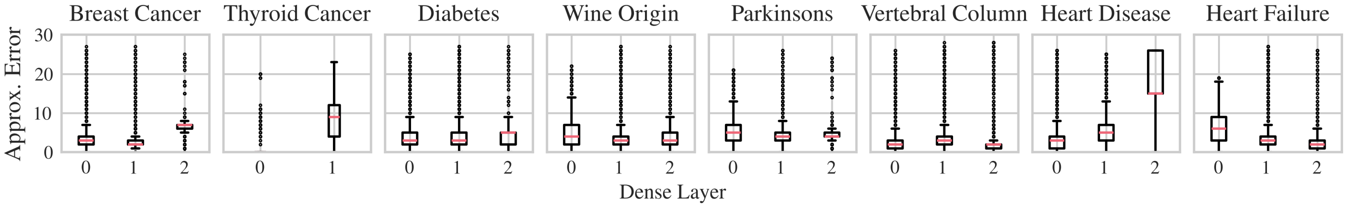


Fig. 6. Approximation error distribution of SILENZIO's approximated block-scaling gadget during the training.

Next, we trained all MLP models with the exact block-scaling implementation for comparison. As shown in Figure 5, the training progress of SILENZIO with the approximated scaling gadget is, despite the partly large approximation errors, very similar compared to training using the exact block scaling implementation. To conclude, we found, that the approximation error of SILENZIO's new block scaling can be extreme on edge cases but is relatively small on average and does not affect the progress of the evaluated MLP training tasks.

## VI. RELATED WORK

In their seminal work, Nandakumar *et al.* [13] proposed the first non-interactive outsourcing scheme for the training of MLPs based on the BGV FHE-scheme. The approach uses 8 bit integer inputs and 16 bit integer weights. They use sigmoid activations and optimize a quadratic loss using SGD during training. The evaluations showed the successful training of a model for the MNIST dataset, achieving 96% accuracy. Measured by today's standards, the scheme is not secure by assuring only 80 bit of security. Unfortunately, the scheme also lacks an openly available implementation.

Glyph [14] proposed by Lou *et al.* introduces a scheme-switching technique between TFHE and BGV to accelerate the non-interactive training of MLPs by performing non-linear activation functions like ReLU and softmax using TFHE and multiply accumulate operations protected through BGV. Further, Glyph demonstrates the fine-tuning of a CNN which brings two performance benefits. First, Glyph heavily reduces the number of trainable parameters; second, operations on the fixed parameters can be performed between ciphertexts and plaintexts, which is generally much faster. Note, that compared to [65], [66], the client does not need to perform the feature extraction locally when using Glyph, making the scheme much more suitable for outsourcing. Glyph leverages a $L^2$ norm loss function implemented in BGV and quantized inputs, weights, and activations using SWALP training quantization [89]. Glyph only achieves a security level of 80 bit and does not provide an openly available implementation.

Montero *et al.* introduced a quantized NN training scheme based on TFHE, which we refer to as TFHE-T [90]. In TFHE-T, after each weight update, the server sends all weights to the client for re-encryption, making the scheme interactive. The authors suggest using bootstrapping operations instead to make the scheme non-interactive, but haven't demonstrated the adaption in an implementation and didn't evaluate it. TFHE-T was configured to achieve a security level of 128 bit. The evaluation demonstrates the training of logistic regression models and MLPs with one hidden layer for the mortality (10 features) and breast-cancer datasets (30 features), achieving near-to plaintext training accuracy. As the authors did not discuss their network settings, it seems they evaluated their interactive approach on a single machine, neglecting the communication overhead over the network, making comparisons to other schemes unfair. To the best of our knowledge, the authors do not provide an open-source implementation.

PrivGD [65] and Hetal [66] are based on the CKKS FHE scheme and provide solutions for non-interactive fine-tuning of a single fully connected layer with an approximated Softmaxfunction on the output. Both schemes require the client to perform local feature extraction on the client-side only partially outsourcing the NN computations. Bourse *et al.* [54] proposed a scheme for non-interactive inference on discretized MLPs. Furthermore, there exists a long line of research studying non-interactive CNN inference [51]–[53], [55]–[60], [62], [64]. Zhang *et al.* [91] presented HEPrune for non-interactive (still requiring light-weight meta-data online-communication) FHE protected data pruning. DataSeal [92] adds integrity to FHE schemes, making them secure against malicious attackers and demonstrated the protection mechanism on NN inference.

Following the latest advancements in natural language processing, Zhang *et al.* [61] proposed non-interactive transformer inference and Panzade *et al.* [67] introduced BlindTuner for non-interactive fine-tuning (using client-side local feature extraction) of a transformer model. Furthermore, there are solutions for k-NN classification [93], tree inference [94], [95], SVM and K-Means classification [96] for the non-interactive outsourcing setting.

## VII. Future Work

Although SILENZIO demonstrates a big step towards secure non-interactive outsourced MLP training, several directions remain open to further improve performance, generality, and practical deployment. To accelerate the FHE execution, Concrete supports compiling for GPU and FPGA targets. Unfortunately, at the time of writing, the compiler backends for CPU and GPU are seemingly unequally mature and our code only compiled successfully for CPU execution. For future research, it would be interesting to also consider hardware accelerators like GPUs and SILENZIO's implementation gives a good starting point, presumably already supporting GPU execution in a future release of Concrete's GPU compiler.

Next to the MLPs supported by SILENZIO, future research should tackle the challenge of supporting more complex architectures, like large CNNs and recurrent models with support for more of the layers used in standard models. SILENZIO's Shift2MSBs$^{\pm}$ block-scaling gadget can also compute the approximated amount of shift performed on the input tensor, this might be helpful to construct fixed-point approximations with higher resolution, similar to NITI [16], required to successfully train larger and more complex model architectures.

Finally, more research is required to further strengthen the security guarantees provided by non-interactive training schemes. To the best of our knowledge, there exists no training scheme providing security against a malicious attacker. In the semi-non-interactive outsourcing setting, [2], [8], [9], [12] achieve security against a malicious attacker, but require non-collusion assumptions. Dataseal [92] provides verifiability for FHE protected computations and demonstrated non-interactive outsourced inference, which could serve as the basis for the development of a training scheme.

## VIII. Conclusion

Based on careful co-design of an approximated MLP training scheme and low-bitwidth integer components efficiently realizable in FHE, we presented SILENZIO. To the best of our knowledge, SILENZIO is the first fully non-interactive framework for outsourcing the training of MLPs, which demonstrates the practical realizability of a 128 bit security guarantee. At its core are three novel building blocks — a low-bitwidth matrix-multiplication gadget, the Shift2MSBs$^{\pm}$ block-scaling mechanism, and an integer-only cross-entropy gradient computation — which together enable a true "fire-and-forget" training paradigm without interaction or non-collusion assumptions. While the non-interactive outsourcing of inference workloads is a well-researched problem, we hope SILENZIO will serve as an impulse for further advancements regarding the less researched and much harder training task.

Our end-to-end implementation in Zama's Concrete library demonstrates that SILENZIO achieves test accuracies on par with standard PyTorch, leveraging 32 bit floating-point computations, while incurring manageable FHE runtime overheads. By removing both client–server interactions and collusion assumptions, SILENZIO offers a drop-in replacement for conventional cloud training services, unlocking non-interactive privacy-preserving outsourced training for sensitive domains such as healthcare and finance. Finally, we will open source SILENZIO's implementation, providing the first open available solution for non-interactive outsourced MLP training.

## Acknowledgment

Generative AI was utilized during programming, editing, and grammar enhancement of this work. This work has been supported by the BMBF through the project AnoMed.

## Appendix A
## RNS-to-MRNS Conversion Example

Table VII shows an example of the conversion process from a number given in RNS representation to the associated MRNS.

TABLE VII
EXAMPLE OF THE RNS TO MRNS CONVERSION PROCESS.

| Moduli $m_1, m_2, m_3$, Radices $r_1, r_2, r_3$ | 5 | 7 | 8 |
|---|---|---|---|
| RNS of $x = 99$ | 4 | 1 | 3 |
| $(x_1, x_2 - 4, x_3 - 4)$ | 4 | 4 | 7 |
| $(x_1, x_2 \cdot 5^{-1} \bmod 7, x_3 \cdot 5^{-1} \bmod 8)$ | 4 | 5 | 3 |
| $(x_1, x_2, x_3 - 5)$ | 4 | 5 | 6 |
| $(x_1, x_2, x_3 \cdot 7^{-1} \bmod 8)$ | 4 | 5 | 2 |
| MRNS of $x = 4 + 5 \cdot 5 + 2 \cdot 7 \cdot 5 = 99$ | 4 | 5 | 2 |

## Appendix B
## Low-bitwidth high-resolution MatMul

Similar to $\mathrm{Matmul}^{\mathrm{RNS}}$, $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$ (see Algorithm 7) gets up to 8 bit matrices as inputs, but the used RNS bases in $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$ consist of larger 5 bit moduli (see Table VIII). The first step is identical in both algorithms. To account for the larger RNS moduli, the second step draws inspiration from the Karatsuba algorithm to keep the intermediate results of the following computations also $\leq$

TABLE VIII
5 BIT RNS BASES.

| $k$ | RNS Base | Max. Bitwidth |
|---|---|---|
| 2 | 31, 30 | 9.86 |
| 3 | 29, 31, 30 | 14.72 |
| 4 | 27, 29, 31, 28 | 19.37 |
| 5 | 25, 27, 29, 31, 28 | 24.02 |
| 6 | 23, 25, 27, 29, 31, 28 | 28.54 |

8 bit and extracts individually the most significant two and least significant three residue-bits of the first input matrix in RNS. In step three, the partial products $\mathbf{Y}_1^*$ and $\mathbf{Y}_2^*$ are computed, where $\mathbf{Y}_1^*$ is immediately reduced modulo the RNS base. To respect the bit extractions of step 2 and perform the modular reduction of $\mathbf{Y}_2^*$ without exceeding 8 bit, we perform the evaluation using a custom lookup table. The last two steps are identical to Algorithm 2, except, that we respect the larger intermediate results and only sum over maximal 7 summands in step 5. Figure 7 shows a step-by-step example of $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$.

**Algorithm 7** $\mathrm{MatmulHighRes}^{\mathrm{RNS}}$

```
Input: $\mathbf{X} \in \mathcal{I}_8^{a\times b}$, $\mathbf{W} \in \mathcal{I}_8^{b\times c}$, $\mathbf{m} \in \mathcal{U}_5^k$    ▷ Inputs, RNS base
Output: $\mathbf{Y} \in \mathcal{J}_5^{k\times a\times b}$    ▷ Output in RNS representation
  ▷ 1. Step: Convert inputs to RNS representation
  $\mathbf{m}' \leftarrow \mathrm{expandDims}(m, [1, 2])$    ▷ Shape: $k \times 1 \times 1$
  $\mathbf{m}'' \leftarrow \mathrm{expandDims}(m, [1, 2, 3])$    ▷ $k \times 1 \times 1 \times 1$
  $\mathbf{X}' \leftarrow \mathbf{X} \bmod \mathbf{m}'$    ▷ $k \times a \times b$
  $\mathbf{W}' \leftarrow \mathbf{W} \bmod \mathbf{m}'$    ▷ $k \times b \times c$
  ▷ 2. Step: Extract bitchunks of input $X'$
  $\mathbf{X}_2' \leftarrow \mathrm{extractBits}(\mathbf{X}', [3, 4])$    ▷ $k \times a \times b$
  $\mathbf{X}_1' \leftarrow \mathrm{extractBits}(\mathbf{X}', [0, 2])$
  ▷ 3. Step: Compute partial products
  $\mathbf{X}_1' \leftarrow \mathrm{expandDims}(\mathbf{X}', 3)$    ▷ $k \times a \times b \times 1$
  $\mathbf{X}_2' \leftarrow \mathrm{expandDims}(\mathbf{X}', 3)$
  $\mathbf{W}' \leftarrow \mathrm{expandDims}(\mathbf{W}', 1)$    ▷ $k \times 1 \times b \times c$
  $\mathbf{Y}_1^* \leftarrow \mathbf{X}_1'\mathbf{W}' \bmod m''$    ▷ $k \times a \times b \times c$
  $\mathbf{Y}_2^* \leftarrow \mathbf{X}_2'\mathbf{W}'$
  $\mathbf{Y}_2^* \leftarrow \mathrm{lookupComp}(\mathbf{Y}_2^* \cdot 8 \bmod m'')$
  ▷ 4. Step: Add partial products
  $\mathbf{Y}^* \leftarrow \mathbf{Y}_1^* + \mathbf{Y}_2^* \bmod m''$
  ▷ 5. Step: Block-wise summation
  $n \leftarrow \min(7, b)$
  $\mathbf{Y} \leftarrow \mathrm{sum}(\mathbf{Y}^*[:, :, : n, :], axis = 2) \bmod \mathbf{m}'$
  for $i \leftarrow n,\ i \le b,\ i \leftarrow i + n$ do
      $j \leftarrow \min(i + n, b)$
      $\mathbf{Y}' \leftarrow \mathrm{sum}(\mathbf{Y}[:, :, i : j, :], axis = 2)$
      $\mathbf{Y} \leftarrow \mathbf{Y} + \mathbf{Y}' \bmod \mathbf{m}'$    ▷ $k \times a \times c$
  end for
```

## REFERENCES

[1] A. P. K. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2183–2200. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

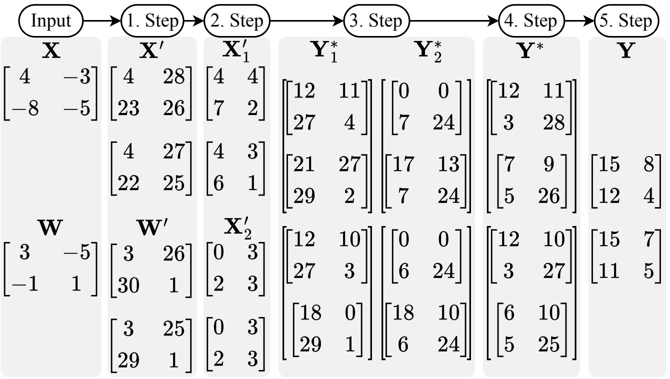


Fig. 7. Minimal step-by-step example of Algorithm 7 for 5-bit RNS-base $[31, 30]$.

[2] N. Koti, M. Pancholi, A. Patra, and A. Suresh, "SWIFT: super-fast and robust privacy-preserving machine learning," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2651–2668. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/koti

[3] J. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 827–844. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/watson

[4] D. Rathee, A. Bhattacharya, D. Gupta, R. Sharma, and D. Song, "Secure floating-point training," 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. 6329–6346. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/rathee

[5] Z. Ren, M. Fan, Z. Wang, J. Zhang, C. Zeng, Z. Huang, C. Hong, and K. Chen, "Accelerating secure collaborative machine learning with protocol-aware RDMA," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/ren

[6] H. Lycklama, A. Viand, N. Küchler, C. Knabenhans, and A. Hithnawi, "Holding secrets accountable: Auditing privacy-preserving machine learning," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/lycklama

[7] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the GPU," in *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 2021, pp. 1021–1038. [Online]. Available: https://doi.org/10.1109/SP40001.2021.00098

[8] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 1, pp. 188–208, 2021. [Online]. Available: https://doi.org/10.2478/popets-2021-0011

[9] N. Attrapadung, K. Hamada, D. Ikarashi, R. Kikuchi, T. Matsuda, I. Mishina, H. Morita, and J. C. N. Schuldt, "Adam in private: Secure and fast training of deep neural networks with adaptive moment estimation," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 4, pp. 746–767, 2022. [Online]. Available: https://doi.org/10.56553/popets-2022-0131

[10] A. N. Baccarini, M. Blanton, and C. Yuan, "Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning," *Proc. Priv. Enhancing Technol.*, vol. 2023, no. 1, pp. 608–626, 2023. [Online]. Available: https://doi.org/10.56553/popets-2023-0035

[11] H. Saleem, A. Ziashahabi, M. Naveed, and S. Avestimehr, "Hawk:

Accurate and fast privacy-preserving machine learning using secure lookup table computation," *CoRR*, vol. abs/2403.17296, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.17296

[12] N. Koti, A. Patra, R. Rachuri, and A. Suresh, "Tetrad: Actively secure 4pc for secure training and inference," in *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/auto-draft-202/

[13] K. Nandakumar, N. K. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 2019, pp. 40–48. [Online]. Available: http://openaccess.thecvf.com/content_CVPRW_2019/html/CV-COPS/Nandakumar_Towards_Deep_Neural_Network_Training_on_Encrypted_Data_CVPRW_2019_paper.html

[14] Q. Lou, B. Feng, G. C. Fox, and L. Jiang, "Glyph: Fast and accurately training deep neural networks on encrypted data," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., 2020. [Online]. Available: https://proceedings.neurips.cc/paper/2020/hash/685ac8cadc1be5ac98da9556bc1c8d9e-Abstract.html

[15] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: fast fully homomorphic encryption over the torus," *J. Cryptol.*, vol. 33, no. 1, pp. 34–91, 2020. [Online]. Available: https://doi.org/10.1007/s00145-019-09319-x

[16] M. Wang, S. Rasoulinezhad, P. H. W. Leong, and H. K. So, "NITI: training integer neural networks using integer-only arithmetic," *IEEE Trans. Parallel Distributed Syst.*, vol. 33, no. 11, pp. 3249–3261, 2022. [Online]. Available: https://doi.org/10.1109/TPDS.2022.3149787

[17] Zama, "Concrete: TFHE Compiler that converts python programs into FHE equivalent," 2022, https://github.com/zama-ai/concrete.

[18] L. K. L. Ng and S. S. M. Chow, "Sok: Cryptographic neural-network computation," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 497–514. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179483

[19] S. U. Hussain, M. Javaheripi, M. Samragh, and F. Koushanfar, "COINN: crypto/ml codesign for oblivious inference via neural networks," in *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, Y. Kim, J. Kim, G. Vigna, and E. Shi, Eds. ACM, 2021, pp. 3266–3281. [Online]. Available: https://doi.org/10.1145/3460120.3484797

[20] S. Balla and F. Koushanfar, "Heliks: HE linear algebra kernels for secure inference," 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. 2306–2320. [Online]. Available: https://doi.org/10.1145/3576915.3623136

[21] R. Lehmkuhl, P. Mishra, A. Srinivasan, and R. A. Popa, "Muse: Secure inference resilient to malicious clients," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2201–2218. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/lehmkuhl

[22] L. K. L. Ng and S. S. M. Chow, "Gforce: Gpu-friendly oblivious and rapid neural network inference," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2147–2164. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/ng

[23] N. Chandran, D. Gupta, S. L. B. Obbattu, and A. Shah, "SIMC: ML inference secure against malicious clients at semi-honest cost," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 1361–1378. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/chandran

[24] Z. Huang, W. Lu, C. Hong, and J. Ding, "Cheetah: Lean and fast secure two-party deep neural network inference," in *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, K. R. B. Butler and K. Thomas, Eds. USENIX Association, 2022, pp. 809–826. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong

[25] F. Liu, X. Xie, and Y. Yu, "Scalable multi-party computation protocols for machine learning in the honest-majority setting," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/liu-fengrun

[26] D. Rathee, A. Bhattacharya, R. Sharma, D. Gupta, N. Chandran, and A. Rastogi, "Secfloat: Accurate floating-point meets secure 2-party computation," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 576–595. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833697

[27] S. Singh, S. Singh, S. Gudaparthi, X. Fan, and R. Balasubramonian, "Hyena: Balancing packing, reuse, and rotations for encrypted inference," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 107–107.

[28] Q. Zhang, T. Xiang, C. Xin, and H. Wu, "From individual computation to allied optimization: Remodeling privacy-preserving neural inference with function input tuning," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2024, pp. 101–101.

[29] K. Gupta, D. Kumaraswamy, N. Chandran, and D. Gupta, "LLAMA: A low latency math library for secure inference," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 4, pp. 274–294, 2022. [Online]. Available: https://doi.org/10.56553/popets-2022-0109

[30] B. Veldhuizen, G. Spini, T. Veugen, and L. Kohl, "Extending the security of SPDZ with fairness," *Proc. Priv. Enhancing Technol.*, vol. 2024, no. 2, pp. 330–350, 2024. [Online]. Available: https://doi.org/10.56553/popets-2024-0053

[31] Q. Zhang, C. Xin, and H. Wu, "GALA: greedy computation for linear algebra in privacy-preserved neural networks," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. [Online]. Available: https://dx.doi.org/10.14722/ndss.2021.24351

[32] C. Dong, J. Weng, J. Liu, Y. Zhang, Y. Tong, A. Yang, Y. Cheng, and S. Hu, "Fusion: Efficient and secure inference resilient to malicious servers," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/fusion-efficient-and-secure-inference-resilient-to-malicious-servers/

[33] K. Gupta, N. Chandran, D. Gupta, J. Katz, and R. Sharma, " Shark: Actively Secure Inference using Function Secret Sharing ," in *2025 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 2472–2490. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP61157.2025.00175

[34] L. Song, J. Wang, Z. Wang, X. Tu, G. Lin, W. Ruan, H. Wu, and W. Han, "pmpl: A robust multi-party learning framework with a privileged party," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 2689–2703. [Online]. Available: https://doi.org/10.1145/3548606.3560697

[35] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "ABY2.0: improved mixed-protocol secure two-party computation," in *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, M. Bailey and R. Greenstadt, Eds. USENIX Association, 2021, pp. 2165–2182. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/patra

[36] Y. Li, Y. Duan, Z. Huang, C. Hong, C. Zhang, and Y. Song, "Efficient 3pc for binary circuits with application to maliciously-secure DNN inference," 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. 5377–5394. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/li-yun

[37] B. Yuan, S. Yang, Y. Zhang, N. Ding, D. Gu, and S. Sun, "MD-ML: super fast privacy-preserving machine learning for malicious security with a dishonest majority," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/yuan

[38] H. Tian, C. Zeng, Z. Ren, D. Chai, J. Zhang, K. Chen, and Q. Yang, "Sphinx: Enabling privacy-preserving online learning over the cloud," in *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2022, pp. 2487–2501. [Online]. Available: https://doi.org/10.1109/SP46214.2022.9833648

[39] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 2023, pp. 63–63.

[40] S. Wagh, "Pika: Secure computation using function secret sharing over rings," *Proc. Priv. Enhancing Technol.*, vol. 2022, no. 4, pp. 351–377, 2022. [Online]. Available: https://doi.org/10.56553/popets-2022-0113

[41] S. Das, S. R. Chowdhury, N. Chandran, D. Gupta, S. Lokam, and R. Sharma, "Communication efficient secure and private multi-party deep learning," *Proc. Priv. Enhancing Technol.*, vol. 2025, no. 1, pp. 169–183, 2025. [Online]. Available: https://doi.org/10.56553/popets-2025-0010

[42] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J. Bossuat, J. S. Sousa, and J. Hubaux, "POSEIDON: privacy-preserving federated neural network learning," in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/poseidon-privacy-preserving-federated-neural-network-learning/

[43] X. Liu, Z. Liu, Q. Li, K. Xu, and M. Xu, "Pencil: Private and extensible collaborative learning without the non-colluding assumption," *CoRR*, vol. abs/2403.11166, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2403.11166

[44] S. Biswas, D. Frey, R. Gaudel, A.-M. Kermarrec, D. Lerévérend, R. Pires, R. Sharma, and F. Taïani, "Low-cost privacy-preserving decentralized learning," *Proceedings on Privacy Enhancing Technologies*, 2025.

[45] C. Harth-Kitzerow, Y. Wang, R. Rajat, G. Carle, and M. Annavaram, "Pigeon: A high throughput framework for private inference of neural networks using secure multiparty computation," *Proceedings on Privacy Enhancing Technologies*, 2025.

[46] J. V. Bulck, F. Piessens, and R. Strackx, "Sgx-step: A practical attack framework for precise enclave execution control," in *Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017*. ACM, 2017, pp. 4:1–4:6. [Online]. Available: https://doi.org/10.1145/3152701.3152706

[47] L. Wilke, F. Sieck, and T. Eisenbarth, "Tdxdown: Single-stepping and instruction counting attacks against intel TDX," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM, 2024, pp. 79–93. [Online]. Available: https://doi.org/10.1145/3658644.3690230

[48] 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

[49] D. Boneh and J. Kim, "Homomorphic encryption for large integers from nested residue number systems," *IACR Cryptol. ePrint Arch.*, p. 346, 2025. [Online]. Available: https://eprint.iacr.org/2025/346

[50] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[51] L. Folkerts, C. Gouert, and N. G. Tsoutsos, "Redsec: Running encrypted discretized neural networks in seconds," in *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023*. The Internet Society, 2023. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/redsec-running-encrypted-discretized-neural-networks-in-seconds/

[52] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016, pp. 201–210. [Online]. Available: http://proceedings.mlr.press/v48/gilad-bachrach16.html

[53] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *CoRR*, vol. abs/1811.09953, 2018. [Online]. Available: http://arxiv.org/abs/1811.09953

[54] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III*, ser. Lecture Notes in Computer Science, H. Shacham and A. Boldyreva, Eds., vol. 10993. Springer, 2018, pp. 483–512. [Online]. Available: https://doi.org/10.1007/978-3-319-96878-0_17

[55] A. Brutzkus, R. Gilad-Bachrach, and O. Elisha, "Low latency privacy preserving inference," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 2019, pp. 812–821. [Online]. Available: http://proceedings.mlr.press/v97/brutzkus19a.html

[56] E. Lee, J. Lee, J. Lee, Y. Kim, Y. Kim, J. No, and W. Choi, "Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 12 403–12 422. [Online]. Available: https://proceedings.mlr.press/v162/lee22e.html

[57] J. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y. Kim, and J. No, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *IEEE Access*, vol. 10, pp. 30 039–30 054, 2022. [Online]. Available: https://doi.org/10.1109/ACCESS.2022.3159694

[58] W. Ao and V. N. Boddeti, "Autofhe: Automated adaption of cnns for efficient evaluation over FHE," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/ao

[59] S. Cheon, Y. Lee, D. Kim, J. M. Lee, S. Jung, T. Kim, D. Lee, and H. Kim, "Dacapo: Automatic bootstrapping management for efficient fully homomorphic encryption," in *33rd USENIX Security Symposium, USENIX Security 2024, Philadelphia, PA, USA, August 14-16, 2024*, D. Balzarotti and W. Xu, Eds. USENIX Association, 2024. [Online]. Available: https://www.usenix.org/conference/usenixsecurity24/presentation/cheon

[60] J. H. Ju, J. Park, J. Kim, M. Kang, D. Kim, J. H. Cheon, and J. H. Ahn, "Neujeans: Private neural network inference with joint optimization of convolution and FHE bootstrapping," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, B. Luo, X. Liao, J. Xu, E. Kirda, and D. Lie, Eds. ACM, 2024, pp. 4361–4375. [Online]. Available: https://doi.org/10.1145/3658644.3690375

[61] J. Zhang, J. Liu, X. Yang, Y. Wang, K. Chen, X. Hou, K. Ren, and X. Yang, "Secure transformer inference made non-interactive," *IACR Cryptol. ePrint Arch.*, p. 136, 2024. [Online]. Available: https://eprint.iacr.org/2024/136

[62] L. Rovida and A. Leporati, "Encrypted image classification with low memory footprint using fully homomorphic encryption," *Int. J. Neural Syst.*, vol. 34, no. 5, pp. 2 450 025:1–2 450 025:16, 2024. [Online]. Available: https://doi.org/10.1142/S0129065724500254

[63] K. Nam, Y. Joo, D. Lee, S. Ha, H. Oh, H. Moon, and Y. Paek, "Lohen: Layer-wise optimizations for neural network inferences over encrypted data with high performance or accuracy," in *Proceedings of the 34th USENIX Security Symposium (USENIX Security '25)*, Seattle, WA, USA, Aug. 2025. [Online]. Available: https://www.usenix.org/system/files/conference/usenixsecurity25/sec24winter-prepub-430-nam.pdf

[64] Y. Ku, F. Liu, C. Hsu, M. Chang, S. Hung, I. Tu, and W. Chen, "Optimizing encrypted neural networks: Model design, quantization and fine-tuning using FHEW/TFHE," *Proc. Priv. Enhancing Technol.*, vol. 2025, no. 4, pp. 1075–1091, 2025. [Online]. Available: https://doi.org/10.56553/popets-2025-0172

[65] C. Jin, M. Ragab, and K. M. M. Aung, "Secure transfer learning for machine fault diagnosis under different operating conditions," in *Provable and Practical Security - 14th International Conference, ProvSec 2020, Singapore, November 29 - December*

*1, 2020, Proceedings*, ser. Lecture Notes in Computer Science, K. Nguyen, W. Wu, K. Lam, and H. Wang, Eds., vol. 12505. Springer, 2020, pp. 278–297. [Online]. Available: https://doi.org/10.1007/978-3-030-62576-4_14

[66] S. Lee, G. Lee, J. W. Kim, J. Shin, and M. Lee, “HETAL: efficient privacy-preserving transfer learning with homomorphic encryption,” in *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 2023, pp. 19 010–19 035. [Online]. Available: https://proceedings.mlr.press/v202/lee23m.html

[67] P. Panzade, D. Takabi, and Z. Cai, “I can’t see it but I can fine-tune it: On encrypted fine-tuning of transformers using fully homomorphic encryption,” *CoRR*, vol. abs/2402.09059, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2402.09059

[68] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=S1_pAu9xl

[69] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, ser. Lecture Notes in Computer Science, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., vol. 9908. Springer, 2016, pp. 525–542. [Online]. Available: https://doi.org/10.1007/978-3-319-46493-0_32

[70] M. Courbariaux and Y. Bengio, “Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1,” *CoRR*, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[71] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 2704–2713. [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/html/Jacob_Quantization_and_Training_CVPR_2018_paper.html

[72] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *CoRR*, vol. abs/1606.06160, 2016. [Online]. Available: http://arxiv.org/abs/1606.06160

[73] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, “Scalable methods for 8-bit training of neural networks,” in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., 2018, pp. 5151–5159. [Online]. Available: https://proceedings.neurips.cc/paper/2018/hash/e82c4b19b8151ddc25d4d93baf7b908f-Abstract.html

[74] S. Wu, G. Li, F. Chen, and L. Shi, “Training and inference with integers in deep neural networks,” in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018. [Online]. Available: https://openreview.net/forum?id=HJGXzmspb

[75] X. Chen, X. Hu, H. Zhou, and N. Xu, “Fxpnet: Training a deep convolutional neural network in fixed-point representation,” in *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*. IEEE, 2017, pp. 2494–2501. [Online]. Available: https://doi.org/10.1109/IJCNN.2017.7966159

[76] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, “Array programming with NumPy,” *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[77] A. Krizhevsky, G. Hinton *et al.*, “Convolutional deep belief networks on cifar-10,” *Unpublished manuscript*, vol. 40, no. 7, pp. 1–9, 2010.

[78] M. A. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: the RPROP algorithm,” in *Proceedings of International Conference on Neural Networks (ICNN’88), San Francisco, CA, USA, March 28 - April 1, 1993*. IEEE, 1993, pp. 586–591. [Online]. Available: https://doi.org/10.1109/ICNN.1993.298623

[79] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.

[80] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[81] W. Wolberg, O. Mangasarian, N. Street, and W. Street, “Breast Cancer Wisconsin (Diagnostic),” UCI Machine Learning Repository, 1993, DOI: https://doi.org/10.24432/C5DW2B.

[82] S. Borzooei, G. Briganti, M. Golparian, J. R. Lechien, and A. Tarokhian, “Machine learning for risk stratification of thyroid cancer patients: a 15-year cohort study,” *European Archives of Oto-Rhino-Laryngology*, vol. 281, no. 4, pp. 2095–2104, 2024.

[83] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, “Using the adap learning algorithm to forecast the onset of diabetes mellitus,” in *Proceedings of the annual symposium on computer application in medical care*, 1988, p. 261.

[84] M. Lichman, “UCI Machine Learning Repository,” https://archive.ics.uci.edu/ml, 2013, irvine, CA: University of California, School of Information and Computer Science.

[85] G. Barreto and A. Neto, “Vertebral Column,” UCI Machine Learning Repository, 2005, DOI: https://doi.org/10.24432/C5K89B.

[86] M. Little, “Parkinsons,” UCI Machine Learning Repository, 2007, DOI: https://doi.org/10.24432/C59C74.

[87] A. Janosi, W. Steinbrunn, M. Pfisterer, and R. Detrano, “Heart Disease,” UCI Machine Learning Repository, 1989, DOI: https://doi.org/10.24432/C52P4X.

[88] “Heart Failure Clinical Records,” UCI Machine Learning Repository, 2020, DOI: https://doi.org/10.24432/C5Z89R.

[89] G. Yang, T. Zhang, P. Kirichenko, J. Bai, A. G. Wilson, and C. D. Sa, “SWALP : Stochastic weight averaging in low-precision training,” *CoRR*, vol. abs/1904.11943, 2019. [Online]. Available: http://arxiv.org/abs/1904.11943

[90] L. Montero, J. Fréry, C. Kherfallah, R. Bredehoft, and A. Stoian, “Machine learning training on encrypted data with TFHE,” in *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics, IWSPA 2024, Porto, Portugal, 21 June 2024*, H. Hu, A. H. Sung, and R. M. Verma, Eds. ACM, 2024, pp. 71–76. [Online]. Available: https://doi.org/10.1145/3643651.3659891

[91] Y. Zhang, M. Zheng, Y. Shang, X. Chen, and Q. Lou, “Heprune: Fast private training of deep neural networks with encrypted data pruning,” in *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, A. Globersons, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. M. Tomczak, and C. Zhang, Eds., 2024. [Online]. Available: http://papers.nips.cc/paper_files/paper/2024/hash/5b26b9e634ba10f6c51c6db7365c4c28-Abstract-Conference.html

[92] M. H. Santriaji, J. Xue, Q. Lou, and Y. Solihin, “Dataseal: Ensuring the verifiability of private computation on encrypted data,” *CoRR*, vol. abs/2410.15215, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2410.15215

[93] M. Zuber and R. Sirdey, “Efficient homomorphic evaluation of k-nn classifiers,” *Proc. Priv. Enhancing Technol.*, vol. 2021, no. 2, pp. 111–129, 2021. [Online]. Available: https://doi.org/10.2478/popets-2021-0020

[94] K. Cong, D. Das, J. Park, and H. V. L. Pereira, “Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 563–577. [Online]. Available: https://doi.org/10.1145/3548606.3560702

[95] R. A. Mahdavi, H. Ni, D. Linkov, and F. Kerschbaum, “Level up: Private non-interactive decision tree evaluation using levelled homomorphic encryption,” 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. 2945–2958. [Online]. Available: https://doi.org/10.1145/3576915.3623095

[96] S. Bian, Z. Zhao, R. Shen, Z. Zhang, R. Mao, D. Li, Y. Liu, M. Waga, K. Suenaga, Z. Guan, J. Hua, Y. Jin, and J. Liu, “CHLOE: loop transformation over fully homomorphic encryption via multi-level vectorization and control-path reduction,” *IACR Cryptol. ePrint Arch.*, p. 1991, 2024. [Online]. Available: https://eprint.iacr.org/2024/1991