REDsec: Running Encrypted Discretized Neural Networks in Seconds

Lars Folkerts, Charles Gouert, Nektarios Georgios Tsoutsos
{folkerts, cgouert, tsoutsos}@udel.edu
University of Delaware

Abstract

Machine learning as a service (MLaaS) has risen to become a prominent technology due to the large development time, amount of data, hardware costs, and level of expertise required to develop a machine learning model. However, privacy concerns prevent the adoption of MLaaS for applications with sensitive data. A promising privacy preserving solution is to use fully homomorphic encryption (FHE) to perform the ML computations. Recent advancements have lowered computational costs by several orders of magnitude, opening doors for secure practical applications to be developed. This work looks to optimize FHE-based private machine learning inference by leveraging ternary neural networks. Such neural networks, whose weights are constrained to {-1,0,1}, have special properties that we exploit in this work to operate efficiently in the homomorphic domain. We introduce a general framework that takes a user-defined model as input (bring-your-own-network), performs plaintext training, and efficiently evaluates private inference leveraging FHE. We perform inference experiments with the MNIST, CIFAR-10, and ImageNet datasets and achieve speeds as fast as 1.6 to 2.2 orders of magnitude slower than unencrypted single-core performance.

I. Introduction

The rapid growth of cloud computing services has amplified concerns about the need for data privacy. Users of these services trust their personal data to the cloud for storage and computation, putting their privacy at risk. For instance, a curious cloud service provider can read the sensitive user data stored on its servers. This allows the provider to learn proprietary secrets as well as personal data (such as health records) to sell to advertisers [1]. In addition, adversaries can mount cyberattacks against cloud servers, exposing private data [2], [3]. Attackers are beginning to set their sights on these servers as more and more users take advantage of cloud services and outsource valuable data. Therefore, direct attacks targeting cloud data centers are becoming increasingly common [4].

These security threats, coupled with the large number of organizations adopting the cloud computing paradigm, make it increasingly necessary to provide security guarantees for outsourced computation. This work concentrates on the specific case of cloud computing security known as machine learning as a service (MLaaS) [5]. In this scenario, a cloud service provider has a trained network with private weights on their servers and allows users to upload their data for classification. For example, cloud service providers can develop and launch novel machine learning algorithms to process medical images using the MLaaS paradigm [6]. However, due to legal and regulatory issues surrounding privacy and IP concerns (e.g., HIPAA [7]), the use of cloud computing for these applications remains constrained. For this scenario to be secure, special considerations need to be in place to enable users to securely upload their data to the cloud and receive provable guarantees about their privacy during processing.

The most common way to secure user data and protect confidentiality is through the use of encryption schemes such as AES [8]. While this successfully prevents attackers and the cloud from viewing the data, it also severely limits the usefulness during processing. Ciphertext data can only be transmitted and stored, and no meaningful computation can be executed on the ciphertext. In other words, common encryption schemes do not allow executing algorithms on encrypted data, such as those required for MLaaS and other cloud computing scenarios [9]. Luckily, the state-of-the-art cryptographic technique called homomorphic encryption (HE) [10] allows for purposeful computation on encrypted data while maintaining confidentiality.

Homomorphic encryption encompasses a particular class of ciphers that share an incredible property: the ability to perform meaningful ciphertext computations which are cryptographically mirrored in the underlying plaintext. This capability allows users to securely outsource computations by sending HE ciphertexts to a third-party cloud service provider. Then the cloud service provider executes algorithms on the ciphertext data without gaining any information about the underlying plaintext. Finally, the cloud sends the encrypted outcome back to the users, and the users decrypt to get the plaintext result. Here HE is the core technology, allowing the user to access the algorithm without revealing her private data and allowing the cloud to keep its algorithm IP unexposed to the user.

Practical HE encryption applications use either leveled HE (LHE) or fully HE (FHE). These types of homomorphic encryption differ in the number and types of operations computed. Many previous works that aim to solve privacy-preserving MLaaS utilize LHE, which only permits arbitrary operations on ciphertext data for *a predefined, bounded number of times* [11]–[13]. One of the fundamental principles of HE is the concept of noise, which accumulates in ciphertexts during every operation and is necessary to guarantee security. If too much noise accumulates in the ciphertexts, the user cannot decrypt the data. LHE abides by this constraint by limiting the depth of LHE computations and requiring the depth to be known beforehand. In essence, LHE schemes need to allocate the proper noise budget ahead of time to ensure correct decryption. In addition, boosting the noise budget is only accomplished by either sacrificing security or progressively increasing the execution time from using larger encryption parameters. For complex algorithms that perform many computations with the same data repeatedly, *LHE does not scale* and becomes incredibly inefficient in terms of both speed and memory overheads. This approach is not feasible for deep neural networks that operate on complex datasets, such as ImageNet. Most LHE works are optimized only for small, straightforward networks for the MNIST dataset [14].

Other works employ FHE, which builds upon LHE and adds a "bootstrapping" mechanism. Bootstrapping allows for noise reduction in ciphertexts when it reaches a certain threshold. This procedure is considered costly and is generally the bottleneck of fully homomorphic operations [15]. However, this approach is still more efficient for complex algorithms than choosing increasingly larger LHE parameters. For example, FHE-DiNN [16] employs FHE to conduct private inference for a tiny neural network, and this approach demonstrates how to evaluate fully-connected layers.

Because LHE realistically only supports inference for neural networks with a small number of layers, in this work we adopt FHE to facilitate inference for *arbitrary* neural networks. While bootstrapping remains the bottleneck of fully homomorphic operations, several works have accelerated the procedure dramatically [15], [17]–[19]. In addition, FHE evaluation can be accelerated even further with GPUs, achieving more than an order of magnitude speedup over a CPU. Even with all of these techniques, the cost of a bootstrap remains much higher than other operations on ciphertexts. Minimizing the invocations of the bootstrap procedure is critical for fast evaluation. We propose a new FHE framework for private neural network inference that is highly configurable and supports GPU acceleration. Contrary to prior works, our REDsec framework employs ternary neural networks (TNNs) and incorporates a strategy known as *bridging* to convert ciphertexts to the integer domain and accelerate arithmetic operations [20]. These insights allow us to reduce the number of bootstraps in a neural network and thus achieve significantly faster inference speeds.

Our contributions can be summarized as follows:

- An optimal order and computation structure of ternary neural networks to accommodate for an efficient fully homomorphic implementation;
- A state-of-the-art (RED)cuFHE library for GPU accelerated HE operations, including leveled operations, encryption of constants, and robust support for multiple GPUs;
- A detailed analysis of neural network structure to determine the most optimal times to perform costly bootstrapping procedures to refresh the noise;

- *Bring your own network (BYON):* An end-to-end system for constructing neural network architectures, including a UI for users to build their model and a compiler to generate training modules in TensorFlow and encrypted inference code in C++/CUDA.

Roadmap: In Section II, we provide an overview of homomorphic encryption and machine learning concepts as well as our adopted threat model. Section III provides an overview of REDsec and Section IV provides specific implementation details. Section V details our experimental evaluations and analysis of results. Lastly, Section VI provides comparisons with prior works and Section VIII concludes the paper.

II. Preliminaries

A. Privacy Preserving Cryptography

This subsection introduces the fundamental concepts for privacy-preserving cryptography. First, we introduce the learning with errors (LWE) problem as the cryptographic foundation of the LHE and FHE schemes. Next, we discuss the classification of HE schemes into partial, leveled, and fully homomorphic capabilities. Following, we enumerate the state-of-the-art HE encryption libraries. Finally, for completeness, we discuss another privacy-preserving technology called multi-party computation (MPC).

1) Learning With Errors (LWE): Learning with Errors (LWE) [21], and its variant called Ring-LWE [22], is the hard problem that many homomorphic encryption schemes and other lattice-based encryption algorithms rely on for their security. In turn, LWE derives its hardness assumptions from other important problems in both coding and lattice theory [23], [24]. Solving the LWE problem requires recovering a function from a set of noisy samples. Adapting this problem to cryptographic applications is relatively straightforward: encryption keys and ciphertexts are injected with noise to hinder cryptanalysis.

2) Types of Homomorphic Encryption: **Partially Homomorphic Encryption (PHE).** The "weakest" category of HE is partial homomorphic encryption (PHE) and was naturally the first realization of homomorphic encryption in general. Cryptosystems such as RSA [25], ElGamal [26], and Paillier [27] fall into this category and allow for only one of two basic arithmetic operations on ciphertexts: either addition or multiplication. These schemes do not derive their security from the previously defined LWE problem, making them fast and suffering from no noise accumulation. However, PHE does not allow for both additions *and* multiplications operations, severely limiting its usefulness. In the context of privacy-preserving machine learning, PHE is often combined with MPC to make up for its computational shortcomings for neural network training and inference [28].

Leveled Homomorphic Encryption (LHE). Contrary to PHE, LHE schemes [11]–[13] allow for arbitrary algorithms to be evaluated on encrypted data because they can support both encrypted additions and multiplications, which form a functionally complete set of operations. However, unlike PHE, LHE cannot execute an unbounded number of encrypted operations because of noise accumulation. This constraint also makes LHE far more challenging to harness since complex encryption parameters must be carefully optimized for both security and the number of operations. The key parameters include the number of primes in the ciphertext moduli chain, the degrees of various polynomials, and the standard deviation of injected noise. Each LHE-based application must attempt to balance the security level, speed, and noise threshold specific to the encryption algorithm. While a similar balancing act exists for FHE, it is generally independent of the actual algorithm. For LHE, the more operations required for an application, the slower (or otherwise less secure) leveled homomorphic operations become in general.

Fully Homomorphic Encryption (FHE). FHE was realized for the first time in 2009 by Craig Gentry with the introduction of the bootstrapping theorem [10]. As mentioned in the introduction, this technique reduces the noise in ciphertexts and thus allows for an arbitrary number of encrypted operations. Without bootstrapping, the only way to eliminate the noise in a homomorphic ciphertext is to re-encrypt the data. In this procedure, a user needs to receive an intermediate ciphertext, decrypt it, re-encrypt it with minimal noise, and finally re-upload it to the server. The bootstrapping procedure converts this concept to the encrypted domain by having the user provide the cloud with *an encryption of the secret key*. The cloud can use this key to perform homomorphic decryption and re-encryption procedures on the ciphertext. The result will be a new ciphertext with significantly reduced noise. Since this procedure is run homomorphically, the plaintext is never exposed to the cloud server. FHE can compute an infinite number of additions and multiplications on ciphertext data by keeping track of noise growth in ciphertexts and applying bootstrapping when needed [19].

3) Contemporary HE Libraries: There are a number of HE libraries to choose from that all offer certain advantages and disadvantages. The first widely available open-source library is called *HElib* [12] and implements the BGV [13] and CKKS [29] homomorphic cryptosystems. The underlying plaintext types are integers (or floating point numbers), and HElib supports both multiplication and addition operations on ciphertext data. This library is frequently used for LHE (even though it supports FHE) because of slow bootstrapping speeds.

Another popular library called *SEAL* [11] was created by Microsoft and offers *leveled* versions of the BFV [30] and CKKS cryptosystems. Similar to HElib, ciphertexts encode either integers or approximate floating-point numbers. SEAL is commonly used to build small privacy-preserving neural network inference applications as it provides an intuitive API and is more friendly and easier to configure than other HE libraries.

Both HElib and SEAL are solid options for LHE, but the former is not practical for FHE, and the latter does not incorporate bootstrapping and thus does not provide FHE support. The *FHEW* [15] scheme, which itself is derived from the GSW cryptosystem [31], takes an entirely different approach to HE from these two libraries and improves the speed of bootstrapping. In FHEW, ciphertexts represent individual bits of plaintext values, and the operations exposed to users take the form of logic gate operations. As a result, algorithms implemented using FHEW must be in the form of (virtual) digital circuits; for instance, to add two encrypted bytes, one must implement an 8-bit homomorphic adder circuit.

While FHEW boasts bootstrapping speeds of less than a second, *TFHE* [19] expands upon and evolves FHEW's approach achieving even more efficient bootstrapping capabilities. Like FHEW, TFHE treats ciphertexts as encryptions of single bits and provides a logic gate API for users to construct arbitrary algorithms as circuits. TFHE can homomorphically evaluate a single boolean gate with bootstrapping in 13 milliseconds and a multiplexer gate with bootstrapping in 26 milliseconds. Due to its incredibly fast bootstrapping speeds, many FHE machine learning frameworks use this library. Our work, REDsec, also utilizes TFHE as the underlying crypto library as it remains the fastest and most feasible option for FHE on CPUs. However, the *cuFHE* [32] and *nuFHE* [33] GPU libraries port the TFHE scheme to CUDA and are capable of accelerating the bootstrapping procedure by over an order of magnitude. To the best of the authors' knowledge, these GPU libraries provide the fastest bootstrapping speeds of any open-source library and boast approximately identical speeds as each other (on the same GPUs and in NTT mode). In its fastest configuration, REDsec employs (RED)cuFHE, which is an major overhaul of cuFHE to evaluate any homomorphic circuit.

4) Multi-Party Computation (MPC): Multi-Party Computation (MPC) involves multiple entities performing functions jointly over their private data. With this approach, no single entity can see the data of other parties involved in the computation. In the context of MLaaS, this means both the cloud and the user share the responsibility for the computation. In practice, several popular private MLaaS solutions incorporate both MPC and HE constructions into their frameworks, including Gazelle [28], Cheetah [34], and MiniONN [35]. These frameworks use either LHE or PHE for linear operations on the cloud (e.g., convolutions) and MPC in the form of garbled circuits [36] for non-linear operations (e.g., ReLU activations). Here MPC can efficiently perform branch decisions, such as the *max* function in ReLU.

MPC Limitations: In these solutions, the cloud still maintains control over the convolution weights, making them transparent to the user. However, *MPC computations actively engage the user*, which limits the usefulness of MLaaS and is infeasible for several applications. In addition, there is a significant communication overhead between the user and cloud as data used for MPC computations must be continuously uploaded and downloaded. Due to these different use cases and constraints, MPC and HE technologies are not directly comparable.

B. Binary Neural Networks (BNNs)

BNNs constrain weights and/or values to {-1,1}. They are primarily researched as a way to store small weight files on mobile devices, as each {-1,1} weight can be represented as a bit {0,1} [37], [38]. In addition to small weight storage, the sign function can be used as the activation function, resetting the values to a single bit. We refer to this class of activation functions as binary activations [37]–[40]. Binary neural networks have many advantages that improve the speed of computation, rendering BNNs less costly in terms of memory and execution time compared to full-precision networks [37]–[40]. Since all weights and values are bits, the bitwise TFHE cryptosystem is ideal for running BNNs.

Several works have expounded on how to train binary neural networks for quick convergence [39]. Training a BNN is an interesting problem since the gradient for the sign function is undefined. Thus, much of the work on BNNs centers around the problem of picking a suitable gradient function during backward propagation for training the network [39]. We remark that our work does not focus on these different implementations, although many are available in the Larq library, which we leverage for implementing REDsec [41].

TABLE I

Popular network architectures for AlexNet: Here we show the weight format, activation functions and reported accuracy. "Mixed" denotes a combination of binary (i.e., sign) and full precision activation functions.

Network	Weights	Activation	Top-1	Top-5
AlexNet [42]	Full Precision	Full Precision	57.1%	80.2%
Binary Weight (BWN) [38]	Binary	Full Precision	56.8%	79.4%
XNOR-net [38]	Binary	Mixed	44.2%	69.2%
BinaryAlexNet [43]	Binary	Binary	36.3%	61.5%
Hybrid Binary (HBN) [44]	Binary	Mixed	48.6%	72.1%
Benn [45]	Binary	Binary	54.3%	N/A

Many other works improve neural network architectures for high accuracy [39], [40], [46]. Indeed, there is a difference between binary weight networks (BWNs) with integer activation functions and binary-weight/binary-activation networks [39], [41]; this trade-off was first explored in the XNOR-net paper [38]. When appropriately trained, binary weight networks can have similar accuracy to full-precision networks. However, binary-weight/binary-activation networks receive some accuracy degradation since information is lost in the binary sign activation function. Likewise, recent works explore hybrid techniques to boost accuracy and still have binary weights and activations [39], [44]; for example, most of the accuracy loss is mitigated by keeping full-precision pixel values at only a few of the middle layers [47]. Another helpful technique is binary network ensembles, where multiple binary neural networks are trained and return a result to the user. In this case, users consolidate these results to improve performance [45]. Finally, RA-BNN networks support early growth, where slowly adding neurons to a BNN helps improve accuracy [46]. Table I compares and summarizes these different strategies.

Ternary neural networks are another promising technique for accuracy improvement, which we find very effective for our work. These networks offer the possibility of having an additional zero weight: {-1,0,1} [48]. This optimization comes with an increase in accuracy, but since it incurs moderate memory and computation overheads, many discrete neural network implementations overlook this feature [39], [48]. However, this cost is effectively negligible when working with encrypted neural networks, making ternary networks a lucrative alternative to binary neural networks for our system.

Due to their low power consumption, BNNs are a practical technique used in edge computing devices. Recent work has used them for emotion detection [49], COVID-19 mask-wearing [50] and Human Activity Recognition [51]. These edge computing devices did not have a threat model, and by deploying the neural network on the device, they put themselves at risk of having the neural network IP stolen [52].

Larq Library: The Larq library for BNNs is actively maintained, integrated into TensorFlow, and is well-documented [41]. Its toolchain supports binary neural network training and has many pre-trained models included in the platform. The API offers QuantDense and QuantConv2D layers for fully connected and 2D convolutions, respectively, in the BNN domain. In addition, it supports many implementations of the sign activation functions, differing in their backward pass pseudo-gradient, as well as ternary [48] and DoReFa discrete activations [53].

C. Threat Model

REDsec is designed with the most common MLaaS scenario in mind: the cloud service provider owns a model and users pay a fee to upload their personal inputs and receive classification results from the cloud. This work is concerned with protecting user data privacy and cloud proprietary network characteristics, such as weights and biases. We assume an honest-but-curious cloud that executes the correct operations on encrypted data but is incentivized to snoop on user data processed by its servers. Likewise, we consider cyberattacks that attempt to exfiltrate sensitive user data from the server. In addition, we assume that the user has limited knowledge about the network architecture and weights and her goal is to protect the privacy of the data send to the cloud.

In terms of user data, the cloud can determine the size and dimensions of the inference inputs. However, encryptions of bits using the TFHE scheme are probabilistic, and operations using encrypted ones and

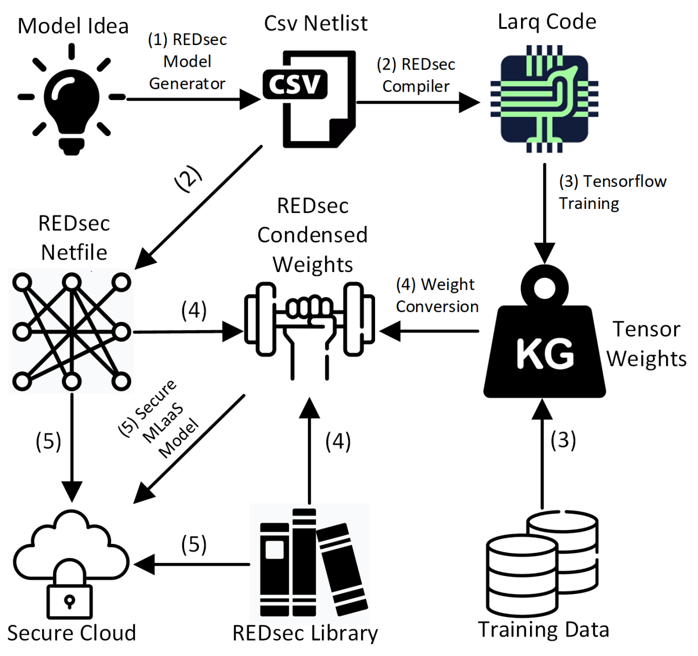


Fig. 1. **REDsec Overview:** Summary and interaction of the different components and modules of the REDsec framework.

zeros take the same amount of time regardless of the underlying plaintext value. Therefore, it is impossible for the cloud to deduce any information about the underlying user data content.

III. Overview of our Methodology

A. End to End System Overview

REDsec is an end-to-end framework that provides an efficient way to generate, train, and execute secure neural network inference for arbitrary, user-defined networks (i.e., *BYON* support). In particular, our framework allows for the execution of configurable networks without writing complex blocks of FHE code. An overview of the REDsec modules is presented here, with references to Fig. 1. A detailed description of implementation details is provided in the next section.

- **REDsec Model Generator (1):** The model generator is a friendly UI tool used to generate a netlist of the neural network model in CSV format. The use of this tool makes our system configurable and easy to use.
- **REDsec Compiler (2):** The REDsec compiler converts the CSV netlists into TensorFlow-based Larq training code and C++/CUDA secure inference net-file code.
- **Training with Larq (3):** Using the Larq library in Jupyter notebook, the model is trained on input data to generate a TensorFlow weights file.

- **REDsec Library:** The C++/CUDA-based REDsec library provides optimizations for efficient, secure inference. These optimizations include reshuffling the network architecture, encryption of circuits, and rigorous parallelism.
- **Weight Conversion with REDsec (4):** Utilizing the REDsec library and generated net-file code, the TensorFlow weight file is condensed and optimized to run using REDsec.
- **Secure Inference (Server) with REDsec (5):** Our server module executes on a remote server. The C++/CUDA net-file code, generated by the REDsec compiler, utilizes the REDsec library and reads the condensed weights file to run secure inference.
- **Outsourced Secure Inference (Client):** REDsec includes client modules to prepare, encrypt and upload the input data, and decrypt the server's results.

B. Secure Inference Overview

REDsec is designed with cloud computing in mind: a remote user communicates with a cloud server, uploads inputs, and receives outputs in turn. REDsec includes client-side scripts that facilitate prepping private inputs, generating encryption keysets, and decrypting classification results. To enable private inference, the cloud and the user both need to initiate separate one-time setup phases. The cloud must specify the neural network and train it. At this stage, the cloud service provider must provide a training set in the clear and a description of the neural network. The cloud can use the REDsec network compiler to generate the corresponding TensorFlow training and REDsec inference code files. After training the network in plaintext and converting the weights, the cloud is ready to receive private inference requests. For the user's setup, the user must generate the HE keypair and send the evaluation key to the cloud to facilitate homomorphic operations.

When the user wishes to classify a private input, she must first supply the data in either picture or binary form to a simple converter module, preparing the input and ensuring that it is in the format that the network is expecting. This may imply simply resizing the image and centering the pixels around the value of 0. Next, the user will utilize an encryption script to take the converted data, encrypt each bit using the private key generated in the setup phase, and export the resulting ciphertext array into a file. The user uploads this file to the cloud, and the cloud initiates the inference procedure.

The cloud will output a ciphertext array, the size of which depends on the network architecture and the number of possible classes in the dataset. For instance, the ImageNet dataset contains 1000 classes, and the result of the inference will be 1000 scores of the input belonging to each class. After encrypted evaluation, the cloud will generate an output ciphertext file encoding the encrypted score of each class, and send this to the user.

Once receiving the output ciphertexts, the user can use their secret key to decrypt the scores to find the class with the maximum classification score. In accordance with other FHE works [16], [54], [55], REDsec opts not to do this in the encrypted domain because encrypted comparator circuits are inefficient and do not scale well with increasing numbers of classes. In addition, the raw scores for each class may provide relevant information to the user, especially when the second-highest score is comparable to the maximum. Therefore, users can sort or find the maximum output, depending on their applications. This is the only computation executed by the user besides basic preprocessing, encryption, and decryption.

C. BNN Optimizations

This section provides an overview of our optimizations. The underlying mathematical formulations are found in the appendix section IX.

1) Multiplication and Data Reuse: One valuable concept in binary neural networks is the limited values that weights can take on. With this realization, REDsec can benefit from performing basic operations once and reusing the result. Specifically, the `XNOR` binary weight multiplications only need to be performed twice per value: once for a $\{$-1$\}$ weight, and once for a $\{$+1$\}$ weight. This technique is specific to REDsec and reduces the multiplications from $O(n_i \cdot n_{i+1})$ to $O(2 \cdot n_i)$, where n is the number of neurons in the layer.

Binary-Binary Multiplication using Univariate `NOT`: Furthermore, REDsec simplifies the binary multiplication problem since the cloud knows the plaintext weights. Therefore, multiplication with weight value $\{$+1$\}$ can utilize the multiplicative identity to eliminate the `XNOR` gate, and multiplication with weight value $\{$-1$\}$ can substitute the noisy bivariate `XNOR` gate with a low noise, univariate `NOT` gate. REDsec stores these encrypted multiplication results in an array and later selects the results according to the plaintext weights. Notably, this applies to both bitwise and integer multiplication. In practice, we combine the `NOT` operation with a bridging procedure to switch the ciphertexts from the binary to the integer domain.

Binary-Integer Multiplication using 2's Complement: REDsec supports integer-value/binary-weight multiplication using a similar logic: multiplication by -1 is the 2's complement, where all bits are flipped and one is added to the result. The bit flip can also be accomplished by a `NOT` gate, while the plus one is rolled into the bias term:

$$-x = \overline{x} + 1. \tag{1}$$

Ternary Weights: For ternary neural networks, multiplication by a 0 weight is 0 regardless of the input. Therefore, we do not need to process the input in our calculations but need to adjust for the zero-valued result in the convolution step [48]. REDsec effectively loads a 0 value in this case, which requires adding $1/2$ offset adjustment to the bias term when switching from the integer {-1,0,1} to the binary {0, 0.5, 1} domain.

Novel Contributions: The reduction of the `XNOR` gate to a `NOT` gate and the reuse of weights is a novel contribution in REDSec. TAPAS [56] performs a different mathematical operation which is not as efficient and does not apply to inputs in the integer domain and does not work with efficient bridging. A mathematical comparison of the REDsec and TAPAS approaches are in the appendix section IX.

2) Activation Functions:

Sign Function and Offset Conversion: The sign function requires a comparison, which is expensive to perform in the encrypted domain. To get around this, we apply an offset value equal to $\text{sign}_{\text{offset}} = 2^{M_{bits}} - M/2$, where M represents the theoretical maximum input value to the activation function, and M_{bits} is the number of bits required to represent M such that $M_{bits} = \lfloor log2(M) + 1) \rfloor$. We end up with the following:

$$bitsign(x) = \begin{cases} +1 & \text{iff } (x + offset) > 2^{M_{bits}}, \\ 0 & \text{otherwise}, \end{cases} \tag{2}$$

which can be simplified by taking the most significant bit of x. Since TFHE operates on bits, and assuming the ciphertext is encoded in the binary domain, the bit extraction is a free operation.

Furthermore, we need to add in multiple offsets throughout the layer. These include:

- 2's complement offsets in integer convolution (eq. 1),
- Ternary zero valued weight offset (sec. III-C1)
- Batch Normalization or Bias offset, and
- Bit sign addition offset (eq. 2).

These offset values can be combined after training to condense the size of the weights file. Combining these offsets also means that during inference, each layer needs to apply only one M_{bit} addition per value. Therefore, even though REDsec's implementation of the activation function requires an addition, there is no additional cost to our activation function since it is combined with other operations. Similar techniques have been applied in other HE works [16], [55]–[57].

ReLU: REDsec also deploys a discretized ReLU function, based on DoReFa-Net discretization [53]. Since the ReLU function is an integer activation, we cannot ignore BatchNorm slope adjustments or overflow detection. Hence, ReLU requires five parts:

1) Batch Norm slope multiplication,
2) Offset Addition,

3) Bridging from the Integer to Binary Domain
4) Right shift for fixed point number consistency,
5) ReLU activation with overflow detection.

First, the slope multiplication is between an encrypted integer and a constant (the BatchNorm slope). We note that the BatchNorm slope output is a floating point number, so adjustments must be made in the final steps of training to ensure discretization of TensorFlow will not affect encrypted neural network output. The derivation of the BatchNorm slope and offset are found in the Appendix.

The offset addition involves the same offsets as in sign activation, except that the bit sign offset is substituted with a discretization offset. This is equal to $1/(2 \cdot M_{bits})$.

The remaining steps are simple. Bridging switches back to the binary domain, so that we can trivially perform the remaining ReLU steps. The shift operation performs the discretization, output bits are determined by the model owner before training. Finally, the ReLU is performed by performing a bitwise-`AND` of the inverse sign bit and the ciphertext output bits. Finally, in correspondence with DoReFa-Net [53], we must clip the output of the activation. Here, an `OR` of the overflow bits must be performed with the ciphertext output bits.

Novel Contributions: For sign activation, the use of a sign offset was discussed in TAPAS [56], but was never implemented. For ReLU activations, REDsec implements the procedure from DoReFa homomorphically. Our ideas of bridging to the binary domain and performing a bitwise ReLU are unique to this work.

3) Pooling Functions:

Max Pooling: Max pooling is not typically used in other homomorphic networks since a costly comparison must be made between values. For REDsec, we can leverage a common BNN technique by moving the max pooling step after the binary activation function, resulting in the transformation $sign(max(\tilde{\mathbf{x}})) = max(sign(\tilde{\mathbf{x}}))$. In this case, the max pooling problem is reduced to an `OR` gate.

Average Pooling: REDsec uses homomorphic-friendly SumPooling in place of AveragePooling. Indeed, prior works leveraging LHE for private neural network inference have used this technique as well [58] [59]. For layers with binary activations, the change is transparent since $sign(avg(\tilde{\mathbf{x}})) = sign(sum(\tilde{\mathbf{x}}))$. For layers with integer activations (e.g., ReLU), we do a multiplication-shift in combination with the BatchNorm multiplication.

Novel Contributions: Replacing AveragePooling with SumPooling is often used with LHE [58], and the `OR` MaxPooling is often used for plaintext BNNs [37]. REDsec is the first to implement these concepts using FHE.

D. FHE Optimizations

The standard open-source TFHE (and cuFHE) implementation assumes that every gate evaluation is bootstrapped (except for the homomorphic `NOT` gate, which essentially results in minimal noise growth). This paradigm is typically referred to as *gate bootstrapping* mode. This mode results in relatively slow homomorphic operations even with the superior bootstrapping capabilities of this scheme, on the order of 10-13 milliseconds for TFHE and 0.5 milliseconds for cuFHE. While this is an impressive result for FHE, it is still prohibitively slow for complex algorithms. Large neural networks require billions of gate evaluations for inference; even small networks require millions of gate evaluations.

Efficient Operations: To compensate for this, we adopt two approaches, namely *bi-directional bridging* and *lazy bootstrapping*. The first technique involves switching from the binary to the integer domain in order to evaluate efficient addition operations; this can be done with a single bootstrapping operation and is simply a matter of *dividing the TFHE torus into more segments*. REDsec uses bridging to minimize noise growth since homomorphic integer addition is relatively low-noise. Bridging also eliminates the high number of bootstraps required to evaluate an addition circuit in the binary domain. This is discussed further in Section IV.

Lazy bootstrapping utilizes homomorphic noise estimations to limit the number of bootstraps and make it easier for the user to implement a neural network. It is well known how additions and multiplications affect noise magnitude [31], and all TFHE operations (in both the binary and integer domain) are composed of additions and multiplications (modulo 2 for the binary case). Therefore, REDsec can accurately estimate the new noise variance after every type of computation on encrypted bits and integers. When the noise level exceeds a threshold, REDsec performs bootstrapping and thus allows for another series of leveled operations. This paradigm is familiar to common-use cases of schemes such as BFV [30] and CKKS [29], but rarely applied to the TFHE cryptosystem.

Lazy Bootstrapping with Noise Auto-tuning: To optimize the lazy bootstrapping approach, we introduce a novel methodology to determine bootstrapping locations on the first inference procedure of a given network. The locations are determined strictly by noise levels or the need to maintain the correctness of binary operations. We note that the TFHE bootstrap is integral to the evaluation of gate operations as it serves to scale the output to the correct region of the torus. In practice, only several gates can be evaluated on a ciphertext before bootstrapping is required to rescale the underlying plaintext value. This mechanism allows us to avoid computing noise levels for each ciphertext after each operation during actual inference, as it only needs to occur during the first live inference computation or on dummy data. Noise auto-tuning adds noise checks after each operation and designates locations in the network code where the noise variance exceeds a certain threshold. In these locations, the noise auto-tuning procedure places a bootstrap procedure. This threshold is a function of the plaintext space; the noise must not exceed the bounds of a torus slice, and the variance can therefore grow to $\frac{1}{2P}$, where P is the plaintext space or alternatively the number of torus slices. After all bootstrap points are determined, there is no need to perform noise checks on subsequent inferences since the noise will grow the same way each time.

(RED)cuFHE: Lastly, we introduce a significant overhaul to the cuFHE library for efficient GPU evaluation of homomorphic circuits. First, the original cuFHE library only supports a single set of parameters corresponding to the recommendations outlined in the original TFHE paper [19]. This parameter set corresponds to 80 bits of security, and while it is a solid default configuration, it is not optimal for all types of applications and algorithms. We introduce changes to the library that allow for customization of injected noise levels and alterations of the LWE polynomial degrees. The latter change influences sizes of both keys and ciphertexts. These improvements allow users to find optimal parameter sets for their particular requirements.

Moreover, we introduce leveled gates (i.e., gates without any bootstrapping), a robust API for leveled integer addition and scalar multiplication, and API extensions for encrypting constant values. These constant values are cryptographically public, enabling the cloud to encode the values with zero noise. This mechanism allows private information from the user *to be mixed with non-private data*. We emphasize that this is a secure operation; assuming one of the operands is a secure ciphertext with noise, the output of a mixed operation will always have a noise level greater than or equal to the highest noise level in one of the operands. In the case of neural network inference, REDsec converts weights and biases into noiseless, encrypted constants to interface with the uploaded inputs encrypted with the secret key.

IV. Description of our Framework

A. Model Generation and Training

In order to make our REDsec framework more accessible, we developed a bespoke compiler for bring-your-own encrypted neural networks. First, the REDsec model generator script generates a CSV netlist. This CSV netlist outlines each neural network layer, listing the convolution dimensions, pooling options, batch normalization, and dropout requirements of the desired network. After the REDsec model generator outputs the CSV netlist, the REDsec compiler can be used on the netlist to generate the training and secure inference code.

For training, the source code output is a Jupyter notebook file that leverages TensorFlow and the Larq library. This notebook file can be executed locally using Jupyter or run on cloud hosts (i.e., one can use Google Colab for debugging or rapid prototyping). After the proper training and validation dataset files are uploaded and linked, the code is ready to train the network. After the network is trained in TensorFlow, a final weight extraction and compression is performed. The REDsec integrated weight converter transforms TensorFlow's floating-point weights to ternary weights and combines different offsets, as discussed in Section III-C2. Finally, the converter outputs a compressed weight file for secure inference.

B. Secure BNN Inference with REDsec

The inference code consists of C++ and CUDA modules that leverage the REDsec, TFHE, and modified (RED)cuFHE libraries. These files are run on the cloud to perform secure inference.

Server Modules: After invoking the compiler in the training step, the cloud service provider should have a compressed weights file and high-level neural network code modules which are fully integrated with the REDsec library. Once the server receives a client's evaluation key, it can begin encrypting the weights and biases as constants (noiseless ciphertexts) prior to receiving secure inputs uploaded by users. After the cloud prepares the encrypted weights and biases, the user can send an inference request and encrypted input data to be classified. The cloud can then execute the generated net code and send the encrypted result back to the user. As it is possible to encode multiple bits in a single ciphertext, the generated net code takes advantage of this to minimize communication overhead and memory consumption by *packing results of classification into a single ciphertext per class*.

Client Modules: Three primary programs are run on the client-side, none of which are computationally demanding. The first is a key generation script that allows users to specify a security level (in bits) and creates a keypair using either TFHE or (RED)cuFHE depending on whether the user wants the cloud to use CPU cores or GPUs for homomorphic computations. The user stores the generated secret key, which the other client-side scripts need for encryption and decryption. The user must also upload the evaluation key directly to the cloud server so the cloud can encrypt the weights, as described previously.

After generating a secure keypair, the user can utilize the second client module to prepare any inputs for secure inference. First, the raw bytes of the image are read, and checks are performed to ensure that the image is compliant with the network architecture. REDsec will ensure that the image dimensions are appropriate for the network, including how many color channels to use. Next, the values of each pixel are encrypted as an array of eight ciphertexts in the binary domain (since the value can vary from 0-255). These ciphertexts are appended to a large ciphertext file that holds all of the encrypted bits of the image. The user uploads this file to the cloud server to start the outsourced secure inference.

When the cloud returns the inference result to the user, the user can use the third client module for decryption. Like the input ciphertext file, the inference results file will vary in size depending on the application and network architecture itself; specifically, it will consist of the number of ciphertexts equal

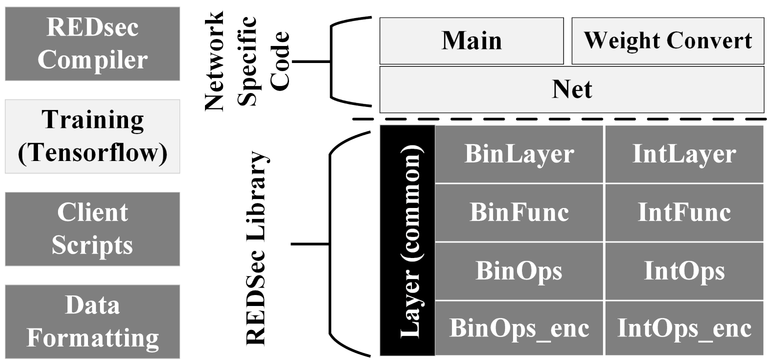


Fig. 2. **REDsec C++/Cuda Library:** This figure shows the REDsec modules, as described in Section IV-C1.

to the number of classes in the dataset. The magnitude of each ciphertext (which is an integer) can grow up to the number of neurons in the last hidden layer, specifically $\lceil \log_2 N \rceil$ where N is the number of neurons. The decryption module will use the secret key to process each ciphertext corresponding to each class and return the plaintext scores. The class with the highest score indicates the most likely match with the input image and can be determined by simply computing the max of the values for all classes.

C. REDsec Library Implementation

The REDsec library contains our implementations of the TFHE machine learning circuits. For additional flexibility, it can be compiled in unencrypted mode for debugging, or encrypted mode for evaluation. This section gives an overview of the library and the optimizations that were contributed.

1) Library Structure: The REDsec library contains the following layers of abstraction, as illustrated in Figure 2:

- **Layer:** The layer library encapsulates convolution, pooling, batch normalization, and activation into a single object. The layer object ensures the proper order of these functions to preserve the REDsec optimizations.
- **Func:** This level of files contains optimized implementations of convolution, fully connected, pooling, batch normalization, and quantize activation functions. OpenMP-based parallelization is added at this level of abstraction.
- **Ops:** The Ops files contain basic, low-level logic and arithmetic circuits that directly invoke the underlying cryptographic library. This level of abstraction has encrypted operation implementations. REDsec supports encrypted operations in CUDA for GPU-based systems and C++ for CPU-based systems.

In addition to these levels of abstraction, we subdivide the functions into integer and binary components based on the layer input. Therefore, the user can use integer layers for higher accuracy or binary layers for speed.

2) Encrypted Circuit Designs: The goal of the homomorphic circuits is to minimize the noise growth in the ciphertexts to delay bootstrapping as long as possible. REDsec attempts to bootstrap only when a conversion between the integer and binary domains is required, thus accomplishing conversion and noise reduction at the same time. This subsection outlines the core building blocks of homomorphic inference used to construct the different network layers.

Adder constructions: We observed that bitwise-adder circuits formed the most computationally expensive operations. For adder circuit designs, the basic building block is a full adder using two `XOR` gates, two `AND` gates, and an `OR` gate. In total, this requires ten arithmetic operations on LWE ciphertexts to evaluate. The high cost of using binary adder circuits is the bootstrapping required to evaluate it successfully. The core problem is that noise accumulates in the carry critical path, leading to more bootstraps and longer latency. Even the carry-lookahead has many bivariate logic in its critical path. Instead, we use a bidirectional bridging technique discussed in section III-D to perform additions in the integer domain. REDsec uses the unique properties of the TFHE bootstrap, *to rescale the ciphertexts* from the binary message space to an integer space (modulo an integer representing the total number of regions on the TFHE torus). This allows us to use the natural FHE addition operation instead of a costly adder circuit composed of logic gates to compute the sum of two ciphertexts. Compared to the naive approach in gate bootstrapping mode that requires five bootstraps per bit of ciphertext, we can accomplish this procedure for the total cost of a single bootstrap for the conversion plus the negligible cost of a single ciphertext addition.

Multiplication: Multiplication circuits are among the slowest to execute in TFHE. However, because REDsec constructs BNNs instead of full-precision networks, the multiplication operation is simply a single homomorphic `NOT` operation and plaintext selector. In the TFHE cryptosystem, this `NOT` operation does not require a bootstrap procedure and becomes among the fastest operations in REDsec networks.

Activation: The most efficient operation in REDsec is the computation of the *sign function*, which is used as the activation function for REDsec networks. While other works that utilize the sign function need to extract the top bit of a ciphertext representing integers, which is an expensive and complicated operation, or perform a programmable bootstrap, we need only make a copy of the ciphertext representing the MSB of the ciphertext vector, assuming the current encrypted value is currently in the binary domain. This operation is fast, accumulates no noise, and is essentially free, which is a major motivation for using BNNs and treating ciphertexts as individual bits for certain operations in the first place.

The discretized ReLU activation function is more involved since the magnitude of the output must be preserved. Because of this, the convolution output must be multiplied by the BatchNorm slope. This is a fairly noisy operation that increases the bit size of the ciphertext, followed by a large-bit addition. Since multiplications and addition circuits are terribly inefficient in the binary domain, we opt to do these operations directly over the integers. Then, we utilize bridging to convert to the binary domain since the remaining operations are exclusively bitwise. We perform a shift operation followed by the ReLU procedure, which can be calculated using a bitwise `AND` with the inverse of the sign bit, which can be extracted for free in the binary domain.

3) GPU Modules for Encrypted Computation: GPUs speed up homomorphic operations significantly over strictly CPU-based systems. For example, a GPU can achieve over a 37x speedup compared to a CPU for bootstrapping operations using the TFHE scheme [33]. For this reason and the fact that cloud instances with GPUs are widely available (such as the P and G families of Amazon EC2 instances), the REDsec library provides GPU support for all homomorphic operations through the use of CUDA code, based on the major (RED)cuFHE optimizations over cuFHE. Notably, through our custom GPU scheduler, (RED)cuFHE can effectively utilize GPU resources for any arbitrary number of available GPUs.

REDsec uses each GPU streaming multiprocessor to handle one homomorphic operation at a time across multiple threads, whereas each CPU core handles one homomorphic operation. While the high degree of parallelization in neural networks helps accelerate plaintext inference, the comparatively high computational cost of homomorphic operations limits the effectiveness of parallelization in the encrypted domain. Instead, we focus on utilizing GPU resources and expand techniques to accelerate expensive homomorphic operations such as bootstrapping, which is the core bottleneck of private inference with FHE.

Updates in (RED)cuFHE: Like the TFHE library, cuFHE only exposes bootstrapped gate functions to users. Even though the bootstrapped operations are much faster on GPUs, they are still orders of magnitude slower than their leveled equivalents. As such, we constructed leveled operations to fit with our lazy bootstrapping paradigm and redesigned the cuFHE ciphertext structures to add variables that track current noise variance. This variable, similarly to the TFHE library, accumulates differently for various operations involving encrypted data. REDsec, therefore, utilizes its *auto-tuning* feature to predict the best places in the network to insert bootstrapping operations. Further, we parameterize (RED)cuFHE to allow for different configurations corresponding to various security levels, as determined using the LWE estimator framework [60].

In addition, the standard cuFHE library only supports HE operations on a single GPU, which limits the amount of potential parallelism. As such, we added the capability to support arbitrary numbers of GPUs regardless of whether or not shared memory is available. Finally, we give cloud servers more control over ciphertext memory transfers between the CPU and GPUs. Each 2-input gate evaluation requires three total memory transfers: two transmissions from the CPU to the GPU for the ciphertext inputs and one transmission from the GPU to the CPU for the ciphertext output. For programs that consecutively operate on ciphertext objects repeatedly (as in the case of BNN inference), this approach is highly inefficient and results in large communication overheads due to the size of HE ciphertexts. With our improvements, cloud servers can transmit encrypted user data at the start of program execution (i.e. prior to inference in the case of REDsec), keep the data on GPUs until all HE operations are complete, and then transmit the output ciphertexts back to the CPU.

GPU resource scheduler: In order to maximize resource utilization for any number of GPUs, we incorporate a custom scheduler to assign GPU streams to CPU threads running inference operations. The scheduler runs on a dedicated CPU thread that maintains an array to keep track of resource utilization and uses shared memory to direct CPU threads to specific GPUs and stream handles. When a CPU thread needs to outsource FHE computation, the scheduler will assign the number of GPU streams proportional to the work required by entering a shared, thread-safe queue. The scheduler will return new stream handles as they become available and when it is at the front of the queue. In the meantime, the CPU thread can utilize its currently assigned hardware resources while it waits for more assignments. When possible, the scheduler will attempt to find resources on a single GPU for any given CPU thread, as communicating with multiple GPUs on a single thread will result in unnecessary overheads. For instance, the CPU thread will need to switch contexts and transfer data back and forth between its assigned GPUs.

V. Experimental Evaluation

To verify and test the efficiency of our framework, we conduct experiments using several network architectures to classify images from three popular datasets at various levels of complexity. We employ a g4dn.metal AWS instance for GPU experiments, which contains eight NVIDIA T4 GPUs (compute capability 7.5) with 40 streaming multiprocessors each and 96 vCPU cores running at 2.5 GHz. For CPU experiments, we use an r5.24xlarge AWS instance, which has 96 vCPUs running at 3.1 GHz. We configure both cuFHE and TFHE for 128 bits of security. The CPU baseline is a single-threaded CPU inference procedure over plaintext data.

We also base our comparisons on the number of multiply-accumulate operations (MACs). MACs influence the complexity of a network as it is the core computational cost of homomorphic inference procedures. As such, MAC costs reflect the efficiency of the inference procedure; the lower the MAC latency, the faster the classification becomes.

TABLE II

SECURE INFERENCE EXPERIMENTS: THE FIVE NETWORKS WERE RAN USING FHE WITH 8 GPUS (GPU-EVAL) AND 96 CPUS (CPU-EVAL) AT 128 BITS OF SECURITY, WHILE PTXT EVAL USES A SINGLE CPU.

	Bin_{1024x3}	ReLU_{1024x3}	BNet_S	BNet	BAlexNet
DataSet	MNIST	MNIST	CIFAR-10	CIFAR-10	ImageNet
Input Size	28x28x1	28x28x1	32x32x3	32x32x3	224x224x3
Activations	Sign	ReLU	Sign	Sign	Sign
Classes	10	10	10	10	1000
Layers	4	4	8	9	8
Int MACs	0	2.3M	1.6M	3.1M	72.9M
Bin MACs	2.3M	0	58.4M	511.0M	768.5M
Bin Weights	2.3M	2.3M	2.0M	10.4M	61.8M
Int Weights	3.1k	6.1k	1.7k	3.9k	11.4k
Ptxt Eval (s)	0.41s	0.41s	2.77	28.3	159
CPU-Eval (s)	12.3	18.4	1081	4622	7472.2
GPU-Eval (s)	3.6	8.2	229	1769.4	5918.5
Accuracy (%)	98.0	99.0	81.9	88.5	61.5 [1]

[1]Top-5 accuracy, as taken from pretrained BYON benchmark [43].

A. *Fully Homomorphic Inference Results*

The REDsec experimental results for various neural network architectures as well as network characteristics are shown in Table II. We preprocess all networks in our experiments using a simple subtraction to center the pixels around 0. We do not include the time required for key I/O and memory allocation when timing the inference procedure. Instead, we compute an amortized cost over five back-to-back inferences after completing this setup phase. Specifically for GPU evaluation, this setup time can take up to a few minutes depending on the complexity of the network in order to allocate pinned memory regions on the host and device memory to prepare for subsequent inferences. However, we remark that this is a one-time cost, and the server can run an arbitrary number of inference procedures in a session without re-allocating memory.

MNIST: For our networks to evaluate the MNIST digit dataset, we use networks called Bin_{1024x1}, Bin_{1024x2}, and Bin_{1024x3}. These networks have have 1,2 and 3 intermediate layers of 1024 neurons respectively (plus the final 10 neuron output layer), all with binary activations. The ReLU_{1024x1}, ReLU_{1024x2}

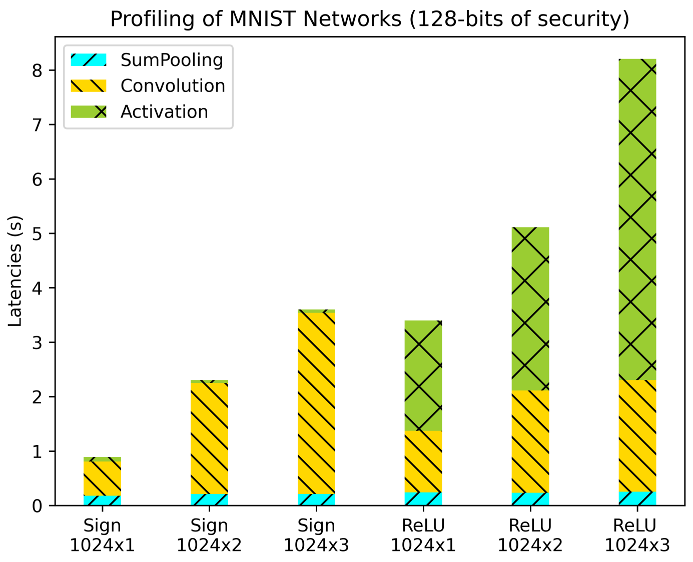
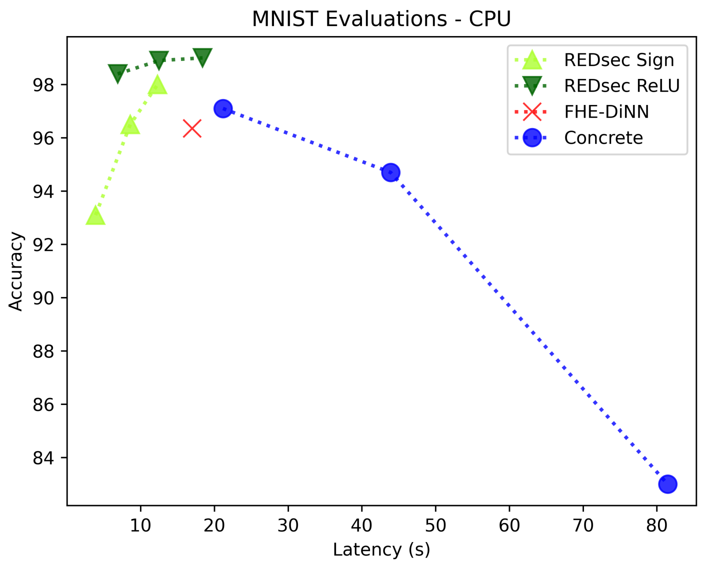


Fig. 3. **Profiling of Performance (128 bits of security):** The timing breakdowns for each MNIST network is shown. ReLU is more expensive than sign, but is found to improve accuracy.

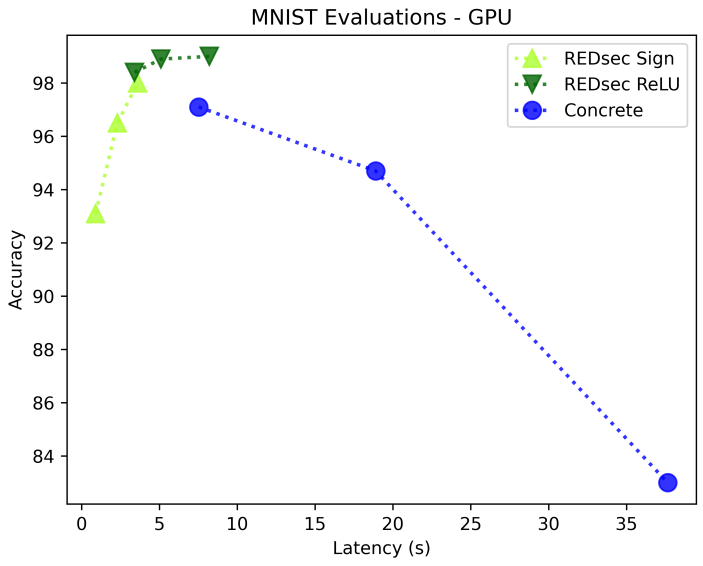
and ReLU$_{1024x3}$ represent identical networks, except with ReLU activations. An encrypted 2x2 SumPool was placed at the beginning of the network to reduce the input size. For the Bin$_{1024x3}$ network, we were able to achieve a 98.0% accuracy for the MNIST dataset. The ReLU networks ave significantly higher accuracy of 99.0%, although they are slower due to an additional bootstrap per neuron attributed to an integer-integer BatchNorm multiplication. The total inference times and the profiling of individual operations are depicted in Figs. 3-4.

CIFAR: We use two architectures to classify the more complex CIFAR-10 dataset, both based on a CIFAR-10 BNN design from BinaryNet [37]. The BNet$_S$ network is a scaled-down version of the original network design and was optimized for speed while maintaining acceptable levels of accuracy. The resulting network uses valid padding and half of the depth of the original BinaryNet architecture in [37]. Notably, this provides an excellent middle-ground benchmark between the relatively simple MNIST network designs and the far more complicated full BNet and AlexNet networks.

ImageNet: For the ImageNet network with 1000 classes, we used a open-sourced pretrained version of BinaryAlexNet for our evaluation [43]. Since REDsec is the first known homomorphic implementation of a network this size, we hope future works can use this common benchmark to compare efficiency across different homomorphic encryption schemes. For this network, we found that despite the memory transfer costs just outlined for GPU evaluation, the BinaryAlexNet evaluation GPU implementation is the closest to plaintext inference speeds, boasting speeds only 37× slower than the unencrypted baseline (Table II).



(a) MNIST CPU Evaluation



(b) MNIST GPU Evaluation

Fig. 4. **MNIST Performance (128 bits of security):** Here we compare accuracy vs. latency for various works and neural network types. REDsec networks can achieve higher accuracy with lower latency than earlier works, while REDsec does not suffer from rounding degradation errors from programmable bootstrapping methods as in Concrete [54]. In addition, REDsec's ReLU_{1024x3} network can run faster than Concrete's NN-100 architecture, even with more than double the estimated multiply-accumulate (MAC) operations.

Overall, REDsec performs well for various network architectures of varying depths and complexity.

VI. Related Works

Secure inference has three different branches of solutions: multi-party computation, pure LHE, and FHE solutions. A comparison of features in relevant secure inference works is summarized in Table IV. A direct comparison of experimental results are shown for FHE solutions in Table III.

Multi-Party Computation Solutions: The first class of secure inference solutions involves multi-party computation [28], [34], [35], [61]–[63]. These schemes send the data back to the user after every layer and require them to perform the non-linear activation function. This model benefits from its ability to apply normal activation functions, such as ReLU or sigmoid. However, this imposes a significant computational overhead on the user, which diminishes one of the motivations for MLaaS: outsourced, offline computation. It also incurs communication overheads of tens to hundreds of megabytes [28], [62]. REDsec also supports non-linear activation functions while allowing for completely outsourced computation on a single server. In addition, REDsec also supports the sign activation function, which is a popular choice for binary neural networks and can evaluate this very efficiently.

Two MPC solutions that use binary neural networks are XONN and Samragh et al. [62], [64]. They propose using the garbled circuit protocol to speed up additions and a comparison-based sign function. Although Samragh et al. exhibit faster runtime and lower communication overhead than XONN, the limitations of residual communication overhead and the requirement of involving the user in the calculation are still present. This is in addition to the other limitations of MPC technology already discussed in Section II.

Leveled Homomorphic Solutions: The second class of secure inference solutions is the pure LHE solution, including the works of CryptoNets [58], Faster CryptoNets [65], n-Graph HE [14] and CryptoDL [59]. These schemes use polynomial activation functions, including the square function x^2 [58], trained quadratic functions $a \cdot x^2 + b \cdot x$ [14] and n-degree polynomials for ReLU, tanh and sigmoid approximations [59]. These approximations either suffer from accuracy loss or generate lots of noise. The polynomial activation functions lack plaintext library support since neural networks need to train on exact functions. In addition, LHE schemes also use a technique called *batching* to improve data throughput. This technique allows users to upload multiple images at a time. However, despite high throughput, the latency is high. Another downside is that one sacrifices any hope of a fast bootstrapping procedure when maximizing the number of batching slots. This is a key reason why batching does not play a large role in FHE evaluation. REDsec, on the other hand, is optimized for low latency through its use of the TFHE library. Most users will not be able to take advantage of the high number of slots that batching provides (typically up to 8192 slots), as this would require an individual user to need thousands of images to be classified at once.

TABLE III

RESULT COMPARISON: MAX MACS CORRESPONDS TO THE LARGEST MODEL EVALUATED ON THE FRAMEWORK IN MILLIONS OF MULTIPLY-ACCUMULATE (MAC) OPERATIONS. REDSEC SCALES TO NETWORKS NEARLY 10X LARGER THAN THE CONCRETE AND SHE FRAMEWORKS AND ORDERS OF MAGNITUDE LARGER THAN FHE-DINN.

Performance	REDsec (this work)		Concrete [54]	FHE-Dinn [16]	SHE [55]
Biggest Dataset	ImageNet		MNIST	MNIST	ImageNet[1]
Solution	FHE		FHE	FHE	LHE
Max MACs (M)	841.3		$\sim$1.0	0.02	28
Weight Size	Micro		Small	Small	Small
Training Lib.	TensorFlow		N/A	Custom	N/A
Input Type	Int		Int	Bin	Int
Weight Type	Tern		Int	Int	Scaled
Activ. Type	Bin		Int	Bin	Int
MNIST (80-bit FHE implementations on CPU)					
Train Time	5 minutes[3]		N/A[2]	11 hours	N/A[2]
Neurons	184	280	1840	100	256
Inference (ms)	2296	3131	2850	1640	3 hours
Accuracy (%)	97.2	98.0	97.2	96.4	99.54

[1]SHE uses ShuffleNet [66], which is an ImageNet architecture one order of magnitude smaller than AlexNet [42].

[2]No known open source implementation.

[3]We use a ReLU_{184x1} and ReLU_{280x1} networks for comparison.

Finally, due to noise accumulation, LHE schemes are not scalable to deeper networks. One can use bootstrapping to make these schemes (e.g., BGV and CKKS) into FHE constructions, but preliminary results in this area have yielded prohibitively slow evaluation times. In the case of CKKS, Lee et al. demonstrated that inference using a ResNet-20 model on the CIFAR-10 dataset takes approximately four hours [67]. REDsec, on the other hand, uses TFHE-based schemes, which have efficient bootstrapping methods and allow for neural network architectures with unlimited depth.

One recent LHE BNN work [68] proposes a three-party threat model for MLaaS. Their system assumes an edge device with private user data, a company server run by the edge device maker, and a cloud server that owns the model. The cloud server divides the model into full-precision and BNN layers. It sends

TABLE IV

COMPARISON OF COMMON NN FEATURES: SUMMARY OF NN FEATURES AVAILABLE INTO THE PRIVACY-PRESERVING DOMAIN.

Features	FHE	Dataset Size	Weight Size	Conv.	Fully Connected	MaxPool	AvgPool	Batch Norm	Dropout	Sign Activ.	ReLU	Int. Inputs	Configurable	Implemented
Cheetah [34]	○	●	○	●	●	◐[1]	○	○	○	○	◐[1]	●	○	●
Gazelle [28]	○	◐	○	●	●	◐[1]	○	○	○	○	◐[1]	●	○	●
SecureML [61]	○	○	○	○	●	○	○	○	○	○	◐[1]	●	○	●
XONN [62]	○	◐	○	◐[1]	◐[1]	◐[1]	○	●	○	○	●	●	○	●
MP2ML [63]	○	○	○	●	●	◐[1]	●	○	●	○	◐[1]	●	○	●
MiniONN [35]	○	◐	○	◐[1]	◐[1]	◐[1]	◐[1]	●	●	○	◐[1]	●	●	●
CryptoNets [58]	○	○	○	●	●	○	●	○	○	○	○	●	○	●
Faster CryptoNets [65]	○	○	○	●	●	○	●	●	○	○	◐[2]	●	○	●
n-GraphHE [14]	○	●	○	●	●	○	●	●	●	○	●	●	●	●
CryptoDL [59]	○	◐	○	●	●	○	●	●	●	○	◐[2]	●	●	●
Concrete [54]	●	○	○	●	●	●	●	●	○	○	●	●	○	●
FHE-Dinn [16]	●	○	●	○	●	○	○	○	○	●	○	○	○	●
Hopfield [57]	●	○	●	○	○	●	○	○	○	●	○	○	○	●
SHE [55]	●	◐	○	●	●	●	○	●	○	○	●	●	○	○
Tapas [56]	●	○	○	●	●	○	○	○	○	●	○	○	○	○
REDsec	●	●	●	●	●	●	●	●	●	●	●	●	●	●

[1]Done by the client using garbled circuits. [2]These operations are polynomial approximations.

the full-precision layers to the edge device company server. The user gives the edge device company server their personal data, which they run unencrypted through their part of the network. The edge device company server transmits the result to the cloud server, which uses HE to run the BNN layers. Their work claims the model is protected since the edge server does not have access to the entire model. However, it assumes that the edge device company server can be trusted with the user's personal data, counterintuitive to privacy-preserving computation. It also requires the edge device company to maintain an expensive server. The REDsec framework can also support this threat model, but practical use cases are more limited due to these drawbacks. In addition, their implementation uses the SEAL library, which only supports LHE. Therefore, they can only run a few HE BNN layers at a time [68].

One other leveled homomorphic network, SHE, differs from the other networks in that it uses TFHE [55]. SHE also includes partial FHE support, but these results are extrapolated and practically infeasible. SHE attempts to implement a full-precision neural network by approximating weights to a power of two. This multiplication optimization allows the circuit to be an `XNOR` followed by a bitshift [55]. Although mixed-precision BNNs sometimes use this technique, it is not yet supported in the current version of the Larq library [39], [41], [53], [55], [69]. The main limitation of SHE is that it builds its architecture upon the boolean logic gate operation in TFHE, which is very slow. This increases inference times compared to using integer arithmetic [55], [70]. Since bootstrapping is not utilized in the leveled mode of operation, the network still suffers from the scalability problems mentioned for other LHE frameworks. For FHE mode, they extrapolate that their framework in gate bootstrapping mode would take over 3 hours to run MNIST inference (our work performs inference in 1.6 seconds instead). While SHE has no working AlexNex implementation, its authors extrapolate that a leveled implementation would take 24.7 hours to run [55], compared to our experimental evaluation of AlexNet that runs in 1.64 hours. Finally, the currently available implementation of SHE covers only basic functional units instead of a complete secure inference neural network [71].

Fully Homomorphic Solutions: The third class is the fully homomorphic branch of solutions, to which REDsec belongs. Compared to other FHE works, and the categories previously explained, REDsec has the most robust support for various types of ML layers as shown in Table IV. It also supports cleartext weights as well as large datasets (indicated by the black circles). Few existing works use the FHE approach, but since the introduction of TFHE with its efficient bootstrapping procedure, several works have proposed unique solutions, as summarized in Tables III and IV. FHE-DiNN [16] uses binary neural networks to achieve inference, and its main innovation is the programmable bootstrapping operation, where a lookup table is used during a bootstrap to evaluate an activation function. While this allows for generic non-linear activations, the sign function can be calculated very efficiently for our scheme in the binary representation. In addition, REDsec optimizes convolution, max pooling, sum pooling, and leveled operations through the use of bridging, all of which FHE-DiNN does not support. Finally, FHE-DiNN is hard to train, not configurable, and relies on a custom Keras implementation instead of an optimized library for BNN training [16], [72]. This lack of support makes it challenging to use in practice.

Another contemporary work called Concrete focuses on developing integer-based functions with programmable bootstrapping [54]. Their sample neural network codebase is not published, making it difficult to compare to our work [73]. Nevertheless, based on the description, their biggest network has around one million multiply-accumulate (MAC) operations (92 inputs·92 outputs·100 layers) [54]. As a comparison, our ReLU_{1024x3} had 2.3 million MAC operations (and a corresponding higher accuracy), and our AlexNet has 841 million MAC operations. Also, our max operation for MaxPooling does not require a bootstrap for standard window sizes, unlike Concrete, and our convolution requires one bootstrap for each input, where Concrete requires two bootstraps for every multiplication operation [54]. Further, the programmable bootstrapping procedure requires low precision, which results in rounding errors; this is evident in the fact that deeper networks with more layers result in less accuracy. For instance, the NN-20 network used for MNIST classification resulted in 97.1% accuracy while the NN-100 (5X more layers) only yielded 83.0% classification accuracy [54]. Figure 4 directly shows this accuracy degradation for Concrete, as well as REDsec's ability to achieve higher accuracy for lower latency.

A third work, TAPAS, presents theoretical approaches to secure inference [56]. While no neural network was implemented, this work extrapolates timing examples from the evaluation on smaller functional units. In addition, this paper did not explore implementation details such as ternary networks, data reuse, integer activations, and bridging. Finally, since there is no material neural network, it remains difficult for future works to expand on their approach.

Finally, a Hopfield network architecture was recently proposed for secure inference [57]. A Hopfield network is a binary recurrent neural network of one or a few layers. It was first proposed in 1982 as one of the first machine learning networks, and some recent applications have made use of them because neurons can run independently in parallel [74], [75]. Given its binary nature, this network architecture is a good choice for secure inference, but it is not as widely used as conventional BNNs for most applications [39], [57]. This solution also uses a small dataset of faces and requires heavy unencrypted preprocessing and feature extraction of facial images by the user. We observe that in terms of multiply-accumulate (MAC) operations, this network is several thousand times smaller than the BinaryAlexNet evaluated in our work.

VII. Future Work

REDsec and other FHE works [16], [54], [55] assume an honest user and an honest-but-curious cloud that will execute the inference procedure correctly but attempt to glean information about user inputs. In future work, we plan to strengthen the current FHE threat model to include malicious users; specifically, we wish to research adversarial machine learning attacks on model IP. Adversarial machine learning attacks can be performed on any MLaaS model, but they require many queries and only produce an accuracy close to the original model [76]–[79]. However, no known work researches adversarial PPML attacks. This would be an interesting research direction since PPML research often does not implement a max function for neural network outputs, which can make adversarial PPML attacks easier. On the contrary, PPML is more costly than current MLaaS services, which may discourage these attacks. Furthermore, binary neural networks are less prone to adversarial attacks [80], [81]. Measuring these tradeoffs and implementing efficient homomorphic defenses provide interesting research problems for future work.

Another future research direction involves including support for residual networks, like ResNet, in our framework to allow for different styles of DNNs and determine the feasibility of FHE for these networks.

VIII. Conclusion

We have presented REDsec, an end-to-end framework for binary and ternary neural network training and secure inference using fully homomorphic encryption. To enable BYON support for our framework, we developed a compiler to output both TensorFlow training and C++/CUDA secure inference code for easy adoption by data scientists and researchers. The secure inference framework exploits binary and ternary network operations to select FHE-friendly functions for convolution, fully connected, max pooling, average pooling, ReLU, and quantization activation layers. Furthermore, we upgraded the cuFHE framework by instituting encryption of constants, leveled operations, and bridging between binary and integer ciphertexts to maximize addition throughput. We demonstrate the capability of our framework by evaluating MNIST, CIFAR-10, and ImageNet, with encrypted speeds only 1.6 to 2.2 orders of magnitude slower than an un-vectorized plaintext CPU implementation.

References

[1] P. Papadopoulos, N. Kourtellis, P. R. Rodriguez, and N. Laoutaris, “If you are not paying for it, you are the product: How much do advertisers pay to reach you?” in *Proceedings of the 2017 Internet Measurement Conference*, 2017, pp. 142–156.

[2] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, “Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds,” in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 199–212.

[3] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Cross-vm side channels and their use to extract private keys,” in *Proceedings of the 2012 ACM conference on Computer and communications security*, 2012, pp. 305–316.

[4] R. Miller, “Foiled plot to attack amazon reflects changing nature of data center threats,” Apr 2021. [Online]. Available: https://datacenterfrontier.com/foiled-plot-to-attack-amazon-reflects-changing-nature-of-data-center-threats/

[5] M. Ribeiro, K. Grolinger, and M. A. Capretz, “Mlaas: Machine learning as a service,” in *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*. IEEE, 2015, pp. 896–902.

[6] A. P. Tafti, E. LaRose, J. C. Badger, R. Kleiman, and P. Peissig, “Machine learning-as-a-service and its application to medical informatics,” in *International Conference on Machine Learning and Data Mining in Pattern Recognition*. Springer, 2017, pp. 206–219.

[7] Centers for Medicare & Medicaid Services, “The Health Insurance Portability and Accountability Act of 1996 (HIPAA),” Online at http://www.cms.hhs.gov/hipaa/, 1996.

[8] NIST, “Advanced Encryption Standard (AES),” in *FIPS PUB 197, Federal Information Processing Standards Publication*, 2001.

[9] A. Sachdev and M. Bhansali, “Enhancing cloud computing security using aes algorithm,” *International Journal of Computer Applications*, vol. 67, no. 9, 2013.

[10] C. Gentry, “Fully homomorphic encryption using ideal lattices,” in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[11] "Microsoft SEAL (release 3.7)," https://github.com/Microsoft/SEAL, Sep. 2021, Microsoft Research, Redmond, WA.

[12] "HElib (version 2.2.1)," https://github.com/homenc/HElib, Oct. 2021.

[13] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[14] F. Boemer, A. Costache, R. Cammarota, and C. Wierzynski, "ngraph-he2: A high-throughput framework for neural network inference on encrypted data," in *Proceedings of the 7th ACM Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2019, pp. 45–56.

[15] L. Ducas and D. Micciancio, "Fhew: bootstrapping homomorphic encryption in less than a second," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 617–640.

[16] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.

[17] C. Gentry, S. Halevi, and N. P. Smart, "Better bootstrapping in fully homomorphic encryption," in *International Workshop on Public Key Cryptography*. Springer, 2012, pp. 1–16.

[18] S. Halevi and V. Shoup, "Bootstrapping for helib," in *Annual International conference on the theory and applications of cryptographic techniques*. Springer, 2015, pp. 641–670.

[19] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachene, "Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds," in *ASIACRYPT*. Springer, 2016, pp. 3–33.

[20] E. Chielle, O. Mazonka, H. Gamil, N. G. Tsoutsos, and M. Maniatakos, "E3: Accelerating Fully Homomorphic Encryption by Bridging Modular and Bit-Level Arithmetic," Cryptology ePrint Archive, Report 2018/1013 v.20200219, 2020, https://eprint.iacr.org/2018/1013/20200219:192239.

[21] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[22] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2010, pp. 1–23.

[23] A. Blum, A. Kalai, and H. Wasserman, "Noise-tolerant learning, the parity problem, and the statistical query model," *Journal of the ACM (JACM)*, vol. 50, no. 4, pp. 506–519, 2003.

[24] S. Khot, "Hardness of approximating the shortest vector problem in lattices," *Journal of the ACM (JACM)*, vol. 52, no. 5, pp. 789–808, 2005.

[25] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978.

[26] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *International conference on the theory and applications of cryptographic techniques*. Springer, 1999, pp. 223–238.

[28] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1651–1669.

[29] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 409–437.

[30] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.

[31] C. Gentry, A. Sahai, and B. Waters, “Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based,” in *Annual Cryptology Conference*. Springer, 2013, pp. 75–92.

[32] W. Dai and B. Sunar, “cufhe (v1.0),” https://github.com/vernamlab/cuFHE, 2018.

[33] NuCypher, “nufhe (v0.0.3),” https://github.com/nucypher/nufhe, 2019.

[34] B. Reagen, W. Choi, Y. Ko, V. Lee, G.-Y. Wei, H.-H. S. Lee, and D. Brooks, “Cheetah: Optimizations and methods for privacy preserving inference via homomorphic encryption,” *arXiv preprint arXiv:2006.00505*, 2020.

[35] J. Liu, M. Juuti, Y. Lu, and N. Asokan, “Oblivious neural network predictions via minionn transformations,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 619–631.

[36] A. C.-C. Yao, “How to generate and exchange secrets,” in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 1986, pp. 162–167.

[37] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1,” *arXiv preprint arXiv:1602.02830*, 2016.

[38] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European conference on computer vision*. Springer, 2016, pp. 525–542.

[39] T. Simons and D.-J. Lee, “A review of binarized neural networks,” *Electronics*, vol. 8, no. 6, p. 661, 2019.

[40] W. Zhao, T. Ma, X. Gong, B. Zhang, and D. Doermann, “A review of recent advances of binary neural networks for edge computing,” *IEEE Journal on Miniaturization for Air and Space Systems*, 2020.

[41] L. Geiger and P. Team, “Larq: An open-source library for training binarized neural networks,” *Journal of Open Source Software*, vol. 5, no. 45, p. 1746, Jan. 2020. [Online]. Available: https://doi.org/10.21105/joss.01746

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[43] Larq, “BinaryAlexNet on Larq Zoo Pretrained Models,” https://docs.larq.dev/zoo/api/literature/\#binaryalexnet.

[44] A. Prabhu, V. Batchu, R. Gajawada, S. A. Munagala, and A. Namboodiri, “Hybrid binary networks: optimizing for accuracy, efficiency and memory,” in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 821–829.

[45] S. Zhu, X. Dong, and H. Su, “Binary ensemble neural network: More bits per network or more networks per bit?” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4923–4932.

[46] A. S. Rakin, L. Yang, J. Li, F. Yao, C. Chakrabarti, Y. Cao, J.-s. Seo, and D. Fan, “Ra-bnn: Constructing robust & accurate binary neural network to simultaneously defend adversarial bit-flip attack and improve accuracy,” *arXiv preprint arXiv:2103.13813*, 2021.

[47] I. Chakraborty, D. Roy, I. Garg, A. Ankit, and K. Roy, “Constructing energy-efficient mixed-precision neural networks through principal component analysis for edge intelligence,” *Nature Machine Intelligence*, vol. 2, no. 1, pp. 43–55, 2020.

[48] F. Li, B. Zhang, and B. Liu, “Ternary weight networks,” *arXiv preprint arXiv:1605.04711*, 2016.

[49] B. Ajay and M. Rao, “Binary neural network based real time emotion detection on an edge computing device to detect passenger anomaly,” in *2021 34th International Conference on VLSI Design and 2021 20th International Conference on Embedded Systems (VLSID)*. IEEE, 2021, pp. 175–180.

[50] N. Fasfous, M.-R. Vemparala, A. Frickenstein, L. Frickenstein, M. Badawy, and W. Stechele, “Binarycop: Binary neural network-based covid-19 face-mask wear and positioning predictor on edge devices,” in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 108–115.

[51] A. De Vita, D. Pau, L. Di Benedetto, A. Rubino, F. Pétrot, and G. D. Licciardo, “Low power tiny binary neural network

with improved accuracy in human recognition systems," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*. IEEE, 2020, pp. 309–315.

[52] X. Hu, L. Liang, L. Deng, S. Li, X. Xie, Y. Ji, Y. Ding, C. Liu, T. Sherwood, and Y. Xie, "Neural network model extraction attacks in edge devices by hearing architectural hints," *arXiv preprint arXiv:1903.03916*, 2019.

[53] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

[54] I. Chillotti, M. Joye, and P. Paillier, "Programmable bootstrapping enables efficient homomorphic inference of deep neural networks," in *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer, 2021, pp. 1–19.

[55] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," *Advances in Neural Information Processing Systems*, vol. 32, pp. 10 035–10 043, 2019.

[56] A. Sanyal, M. Kusner, A. Gascon, and V. Kanade, "Tapas: Tricks to accelerate (encrypted) prediction as a service," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4490–4499.

[57] M. Izabachène, R. Sirdey, and M. Zuber, "Practical fully homomorphic encryption for fully masked neural networks," in *International Conference on Cryptology and Network Security*. Springer, 2019, pp. 24–36.

[58] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *International Conference on Machine Learning*. PMLR, 2016, pp. 201–210.

[59] E. Hesamifard, H. Takabi, and M. Ghasemi, "Cryptodl: towards deep learning over encrypted data," in *Annual Computer Security Applications Conference (ACSAC 2016), Los Angeles, California, USA*, vol. 11, 2016.

[60] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[61] P. Mohassel and Y. Zhang, "SecureML: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2017, pp. 19–38.

[62] M. S. Riazi, M. Samragh, H. Chen, K. Laine, K. Lauter, and F. Koushanfar, "{XONN}: Xnor-based oblivious deep neural network inference," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 1501–1518.

[63] F. Boemer, R. Cammarota, D. Demmler, T. Schneider, and H. Yalame, "Mp2ml: a mixed-protocol machine learning framework for private inference," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–10.

[64] M. Samragh, S. Hussain, X. Zhang, K. Huang, and F. Koushanfar, "On the application of binary neural networks in oblivious inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4630–4639.

[65] E. Chou, J. Beal, D. Levy, S. Yeung, A. Haque, and L. Fei-Fei, "Faster cryptonets: Leveraging sparsity for real-world encrypted inference," *arXiv preprint arXiv:1811.09953*, 2018.

[66] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.

[67] J.-W. Lee, H. Kang, Y. Lee, W. Choi, J. Eom, M. Deryabin, E. Lee, J. Lee, D. Yoo, Y.-S. Kim *et al.*, "Privacy-preserving machine learning with fully homomorphic encryption for deep neural network," *arXiv preprint arXiv:2106.07229*, 2021.

[68] W. Qiang, R. Liu, and H. Jin, "Defending cnn against privacy leakage in edge computing via binary neural networks," *Future Generation Computer Systems*, vol. 125, pp. 460–470, 2021.

[69] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "Fp-bnn: Binarized neural netwfork on fpga," *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.

[70] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, "Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2021, pp. 670–699.

[71] Q. Lou and L. Jiang, "She: A fast and accurate deep neural network for encrypted data," https://github.com/safednn/SHE, 2019.

[72] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," https://github.com/mminelli/dinn, 2017.

[73] Zama, "Zama concrete v0.1.0," https://github.com/zama-ai/concrete, 2021, zama AI, France.

[74] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[75] X. Sui, Q. Wu, J. Liu, Q. Chen, and G. Gu, "A review of optical neural networks," *IEEE Access*, vol. 8, pp. 70 773–70 783, 2020.

[76] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction {APIs}," in *25th USENIX security symposium (USENIX Security 16)*, 2016, pp. 601–618.

[77] H. Yu, K. Yang, T. Zhang, Y.-Y. Tsai, T.-Y. Ho, and Y. Jin, "Cloudleak: Large-scale deep learning models stealing through adversarial examples." in *NDSS*, 2020.

[78] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, 2017, pp. 506–519.

[79] J. R. Correia-Silva, R. F. Berriel, C. Badue, A. F. de Souza, and T. Oliveira-Santos, "Copycat cnn: Stealing knowledge by persuading confession with random non-labeled data," in *2018 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2018, pp. 1–8.

[80] A. Galloway, G. W. Taylor, and M. Moussa, "Attacking binarized neural networks," *arXiv preprint arXiv:1711.00449*, 2017.

[81] M.-R. Vemparala, A. Frickenstein, N. Fasfous, L. Frickenstein, Q. Zhao, S. Kuhn, D. Ehrhardt, Y. Wu, C. Unger, N.-S. Nagaraja *et al.*, "Breakingbed: Breaking binary and efficient deep neural networks by adversarial attacks," in *Proceedings of SAI Intelligent Systems Conference*. Springer, 2021, pp. 148–167.

[82] R. Ding, T.-W. Chin, Z. Liu, and D. Marculescu, "Regularizing activation distribution for training binarized deep networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 408–11 417.

IX. Appendix: DiNN Mathematical Foundations

In this appendix, we present the mathematical foundation of our discretized neural networks. Many of these concepts are taken from generic binary neural networks; our contributions in REDsec are explicitly labeled.

Binary-Integer Mappings: Before delving into mathematical operations, it is important to understand the mappings between the binary and integer domains. For binary inputs in the convolution layer, such as after the sign activation function, there are two possible integer values {-1,1} and three possible weights values {-1,0,+1}. Realizing that zero is a special case, we can map:

$$int \mapsto bin \tag{3}$$

$$-1 \mapsto 0 \tag{4}$$

$$+1 \mapsto 1 \tag{5}$$

and for ternary weights:

$$0 \mapsto 0.5. \tag{6}$$

Ternary weights are added to the bias term, as explained below. Integer inputs are taken at face value, and no special mapping is required.

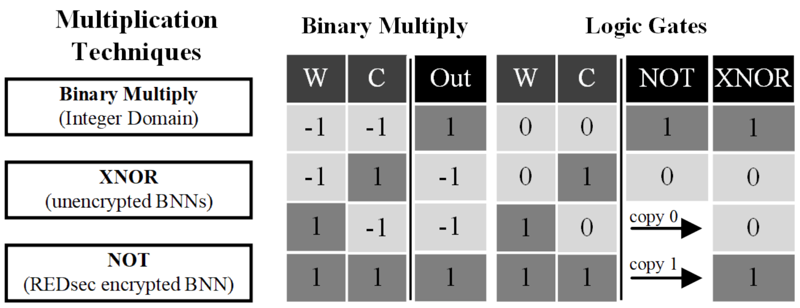
Multiply Accumulate: Multiply-accumulates (MACs) are used in convolution and fully-connected layers. They are described in section III-C1, and typically computed using an `XNOR` gate. This is illustrated in Figure 5. For binary *input* MACs, REDsec converts the XNOR into a homomorphic-friendly `NOT` gate. Zero weights are mapped to 0.5. Since any multiplied by zero is zero, we add this 0.5 to the bias as part of weight preprocessing. *For binary input layer, we use z to denote the zero part of the offset term.* Integer *input* MACs also utilize the NOT gate by exploiting 2's complement. Equation 1 was defined as:

$$-1 * x = \overline{x} + 1. \tag{7}$$

Multiplication by 1 is itself, and multiplication with 0 is 0 (so no operation needs to be performed. *for integer input layer, we instead use z to denote the accumulated 2's complement part of the offset term.*

BatchNorm: BatchNorm is useful in training to recenter the binary neural networks around 0 with a standard deviation of 1. This function is essential for binary neural networks and is studied for BNNs in Ding et. al. [82]. BatchNorm is placed before the activation function, and take the form of

$$BN(x+z) = \frac{(x+z)-\mu}{\sqrt{\sigma^2+\epsilon}} \cdot \gamma + \beta \tag{8}$$


Multiplication Techniques
Binary Multiply
(Integer Domain)
XNOR
(unencrypted BNNs)
NOT
(REDsec encrypted BNN)

Binary Multiply			Logic Gates			
W	C	Out	W	C	NOT	XNOR
-1	-1	1	0	0	1	1
-1	1	-1	0	1	0	0
1	-1	-1	1	0	copy 0 $\rightarrow$	0
1	1	1	1	1	copy 1 $\rightarrow$	1

(a) Strategies (b) Truth Tables

Fig. 5. **Integer to Logical Space:** The core idea behind BNNs is that a complex multiplication operation reduces to an `XNOR gate`, with the {-1,1} in the integer domain mapping to {0,1} in the `XNOR` truth table. In the encrypted domain, when the weight (W) is known to the server, REDsec applies either the `NOT` gate (if W=0) or a copy operation (if W=1) to the ciphertext (C) instead of the noisy and expensive homomorphic `XNOR` gate.

where μ is the output mean during training, σ is the output standard deviation during training, ϵ is a small amount to prevent division by 0, and β is a learned bias offset. This can be coverted into the form

$$BN(x) = m \cdot x + b \tag{9}$$

$$m = \frac{\gamma}{\sqrt{\sigma^2 + \eta}} \tag{10}$$

$$b = \beta - \mu \cdot \frac{\gamma}{\sqrt{\sigma^2 + \eta}} + \frac{z}{\sqrt{\sigma^2 + \eta}} \tag{11}$$

Addition scaling factors are multiplicative applied to the m and b to adjust for the mappings. For binary inputs, this scaling factor is always $1/2$; an increase of 1 in integer domain corresponds to an increase in 0.5 in binary, according to Equations 3-5. For integer inputs, it varies based on the input size; tensorflow maps from -1 to +1, but out network may map from -255 to +255.

For binary *outputs* (i.e., sign), the only the sign of the slope needs to be taken into account since $sign(mx) = sign(m) * sign(x)$. REDsec adjusts for negative slopes in the following layer. For integer *outputs* (i.e., ReLU), the slope m needs to be utilized. We scale m to a user defined integer (we use 8-bits in our experiments). Note that additional training must be run, freezing BatchNorm weights for and allowing the network to adjust to the rounded integer values. This technique is common to most BNN networks [82].

Activation: The sign binary output activation function is straightforward. For binary-input binary-outputs the REDsec weight preprocessor adjusts the bias b from Equation 9 to map 0 in the integer domain to a power of 2 in the binary domain such that the MSB is 1 for all positive numbers. Then REDsec binarizes the ciphertext to extract individual bits, and performs a bitshift to extract the most significant bit, representing positive numbers. For integer-input binary-output, a similar process followed, but the ciphertext is already centered around 0 and REDsec just needs to extract the sign bit. Here, bridging to the binary domain and bit extraction techniques are novel to REDsec.

The discretized ReLU function is more complex. The TensorFlow implementation follows the form of

$$ReLU_{DoReFa}(x) = \begin{cases} 0 & x < \dfrac{1}{2n} \\ \dfrac{i}{n} & \dfrac{2i-1}{2n} < x < \dfrac{2i+1}{2n} \\ 1 & x > \dfrac{2n-1}{2n} \end{cases} \tag{12}$$

where i is the output discretized value, and n is the max value $n = 2^{q_{bits}} - 1$. The output is a clipped ReLU, with outputs rounded to the nearest $\dfrac{1}{n+1}$. To accomplish this homomorphically, REDsec first applies batch norm through Equation 9 by multiplying by the slope m and adding the bias offset b. Then REDsec bridges to get a bit representation of the ciphertext. These bits are shifted to rescale after the multiplication and discretize the output. Then a ReLU is performed by AND-ing with the inverse of the sign bit. Finally, we use OR gates to clip the output values to 1.

Further Comparisons with TAPAS: TAPAS utilizes a different integer-to-binary mapping than REDsec. Instead of a mapping, they utilize a +1 trick that essentially maps $-1 \mapsto 0$ and $+1 \mapsto 1$. This approach requires a bitshift after addition and TAPAS proposes using binary adders to accomplish this. REDsec, on the other hand, utilizes the efficient `NOT` gate for multiplication and bridges to the integer domain for addition. REDsec found this bridging technique to be far more efficient and scalable to wider bit networks than use of binary adders, with many gate operations and noise accumulation in the carry bits.