REDsec: Running Encrypted Discretized Neural Networks in Seconds

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

***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. In this work, we introduce the REDsec framework that optimizes 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 to operate efficiently in the homomorphic domain. REDsec introduces novel features, including a new data re-use scheme that enables bidirectional bridging between the integer and binary domains for the first time in FHE. This enables us to implement very efficient binary operations for multiplication and activations, as well as efficient integer domain additions. Our approach is complemented by a new GPU acceleration library, dubbed (RED)cuFHE, which supports both binary and integer operations on multiple GPUs. REDsec brings unique benefits by supporting user-defined models as input (bring-your-own-network), automation of plaintext training, and efficient evaluation of private inference leveraging TFHE. In our analysis, we perform inference experiments with the MNIST, CIFAR-10, and ImageNet datasets and report performance improvements compared to related works.**

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 [63]. In addition, adversaries can mount cyberattacks against cloud servers, exposing private data [74], [91]. Attackers are beginning to set their sights on these servers as more users take advantage of cloud services. Therefore, direct attacks targeting cloud data centers are becoming increasingly common [56].

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 known as machine learning as a service (MLaaS) [73]. 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 [82]. However, due to legal and regulatory issues surrounding privacy and intellectual property (e.g., HIPAA [9]), 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 [58]. While this successfully prevents attackers and the cloud from viewing the data, it also 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 [76]. Luckily, the state-of-the-art cryptographic technique called homomorphic encryption (HE) [28] allows for 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 it to get the plaintext result. Here HE is the core technology, allowing the user to access the algorithm without revealing her private data, while allowing the cloud to process the data without actually accessing it.

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* [8], [34], [79]. One of the fundamental principles of HE is the concept of noise, which accumulates in ciphertexts

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24034
www.ndss-symposium.org

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 by 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 [5].

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 [21]. However, this approach is still more efficient for complex algorithms than choosing increasingly larger LHE parameters. For example, FHE-DiNN [7] 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 [12], [21], [29], [33]. In addition, FHE evaluation can be accelerated even further with GPUs, achieving more than an order of magnitude speedup over a CPU. Even with these techniques, the cost of a bootstrap remains higher than other HE operations, and minimizing the invocations of the bootstrap procedure is critical for fast evaluation. To mitigate these performance challenges, 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 provides a strategy to enable *bidirectional bridging* to convert ciphertexts to the integer domain for accelerating arithmetic operations, before returning to the binary domain for bitwise operations. 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:

- A new design methodology for FHE-friendly binary and ternary neural network inference, leveraging bespoke multiplication, pooling and data reuse to enable efficient bridging.
- Bidirectional bridging for the first time in TFHE, which enables us to use both efficient binary multiplication and sign extraction operations, as well as integer additions.
- 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 technical details, while Section V discusses our experimental evaluation and analysis of results. Lastly, Section VI provides comparisons with prior works and Section VIII concludes the paper.

II. Preliminaries

A. Privacy Preserving Cryptography

Here we introduce fundamental concepts for homomorphic encryption. First, we discuss the learning with errors (LWE) problem as the cryptographic foundation of LHE and FHE schemes. Next, we discuss partial, leveled, and fully homomorphic HE schemes, and 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): Both LWE [70], [71] and Ring-LWE [54] are hard problems 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 [3], [40]. Solving the LWE problem is akin to decoding from a random linear code [70], [71]. 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).** This is the most basic HE class and was naturally the first realization of homomorphic evaluation. Cryptosystems such as RSA [75], ElGamal [22], and Paillier [61] 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. This allows them to be fast, have no noise accumulation and execute an unbounded number of encrypted operations. However, PHE does not allow for both addition *and* multiplication 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 [39].

Leveled Homomorphic Encryption (LHE). Contrary to PHE, schemes in this category [8], [34], [79] 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). This powerful technique was realized for the first time in 2009 by Craig Gentry with the introduction of the bootstrapping theorem [28]. 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 [12].

3) Contemporary HE Libraries: There are a number of HE libraries which each offer advantages and disadvantages. IBM's *HElib* [34] and Microsoft's *SEAL* [79] implement the BGV [8] and CKKS [11] homomorphic cryptosystems (SEAL also supports BFV [23]). The underlying plaintext types are integers (or floating point numbers), and both multiplications and additions are possible on ciphertexts. These libraries are used for LHE; it is possible to use HElib in FHE mode, but the bootstrapping speeds are prohibitively slow. 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* [21] scheme, which itself is derived from the GSW cryptosystem [30], 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* [12] expands upon and evolves FHEW's approach achieving even more efficient bootstrapping capabilities. TFHE can homomorphically evaluate a single gate with bootstrapping in 13 milliseconds, and due to these 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. Moreover, the *cuFHE* [18] and *nuFHE* [59] 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 offer the fastest bootstrapping speeds of any open-source library so far. In its fastest configuration, REDsec employs our new (RED)cuFHE library, which is an major overhaul of cuFHE to evaluate any homomorphic circuit.

4) Multi-Party Computation (MPC): This technology 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. Moreover, some popular private MLaaS solutions incorporate both MPC and HE constructions into their frameworks, including Gazelle [39], Cheetah [69], and MiniONN [51]. These frameworks use either LHE or PHE for linear operations on the cloud (e.g., convolutions) and MPC in the form of garbled circuits [87] 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 methods, 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 practicality of MPC in "fire-and-forget" style MLaaS 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} and are used as a way to store small weight files on mobile devices, as each {-1,1} weight can be represented as a bit {0,1} [17], [68]. Additionally, the sign function can be used as the activation, resetting the values to a single bit. We refer to this class of activation functions as binary activations [17], [68], [80], [92]. Binary neural networks have many advantages that decrease latency, rendering BNNs less costly in terms of memory and execution time compared to full-precision networks [17], [68], [80], [92]. Since all weights and values are bits, the TFHE cryptosystem is ideal for running BNNs.

Several works have expounded training binary neural networks for quick convergence [80]. Training a BNN is an interesting problem since the gradient for the sign function is undefined. Thus, much work on BNNs centers around picking a suitable gradient function during backward propagation for training the network [80]. We remark that our work does not focus on these different implementations, although many are available in the Larq library [27], which we leverage for implementing REDsec. Larq also has recommendations for high accuracy training [46], [47].

Many other works improve neural network architectures for high accuracy [67], [80], [92]. Indeed, there is a difference between binary weight networks (BWNs) with integer activation

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) [68]	Binary	Full Precision	56.8%	79.4%
XNOR-net [68]	Binary	Mixed	44.2%	69.2%
BinaryAlexNet [45]	Binary	Binary	36.3%	61.5%
Hybrid Binary (HBN) [65]	Binary	Mixed	48.6%	72.1%
BENN [94]	Binary	Binary	54.3%	N/A

functions and binary-weight/binary-activation networks [27], [80]; this trade-off was first explored in the XNOR-net paper [68]. 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 [65], [80]; for example, most of the accuracy loss is mitigated by keeping full-precision pixel values at only a few of the middle layers [10], [62]. 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 [94]. Finally, RA-BNN networks support early growth, where slowly adding neurons to a BNN helps improve accuracy [67]. Table I compares and summarizes these different strategies.

Ternary neural networks offer 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} [49]. 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 [49], [80]. 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 also practical for edge computing devices. Recent work has used them for emotion detection [1], COVID-19 mask-wearing [24] and Human Activity Recognition [19]. However, these edge computing applications 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 [37].

Larq Library. The Larq library for BNNs is actively maintained, integrated into TensorFlow, and is well-documented [27]. Its toolchain supports BNN training and includes many pre-trained models. 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 [49] and DoReFa discrete activations [93].

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 direct access to 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. This threat model does not consider adversarial attacks by the user; it is an open research problem, discussed in Section VII.

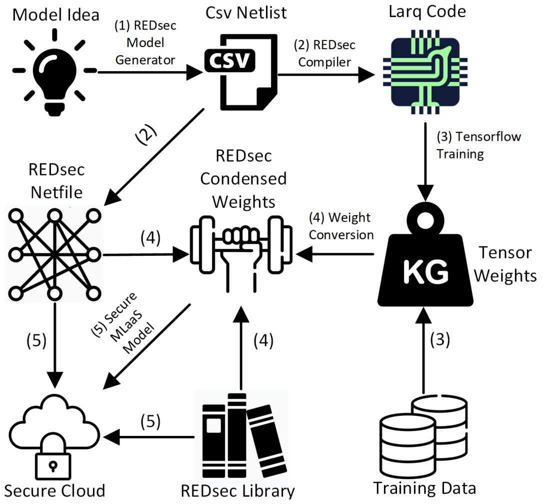


Fig. 1. **REDsec Overview:** Summary and interaction of the different components and modules of the REDsec framework.

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 zeros take the same amount of time regardless of the underlying plaintext value. Therefore, it is imfeasible 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 technical 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. Leveraging 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 netfile code.
- **Training with Larq (3).** Using the Larq library in a Jupyter notebook, the model is trained on input data to generate a TensorFlow weights file.
- **REDsec Library.** Our C++/CUDA-based 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 modules that facilitate prepping private inputs, generating keysets, and decrypting classification results. To enable private inference, the cloud and 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 network architecture. 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 can receive private inference requests. For client 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 converter & verification module, preparing the input and ensuring that it is in the format that the network is expecting. This may entail simply resizing the image and centering the pixels around the value of 0. Next, the user will utilize our encryption module 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.

After 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 [7], [13], [52], 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 presented in the Appendix.

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 our 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 [49]. 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. With sign activations, the $1/2$ offset in bias is rounded using the floor function during weight prepossessing; with ReLU activations, ciphertexts are scaled up to a fixed point precision before adding the bias.

Novel Contributions. The reduction of the `XNOR` gate to a `NOT` gate and the reuse of weights is a key contribution in REDsec, which will enable efficient bidirectional bridging (to be discussed further in Section IV-C2). TAPAS [78] 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 comparison between the REDsec and TAPAS approaches is presented in the Appendix.

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:

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

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. Related techniques have been applied in earlier HE works [7], [38], [52], [78].

ReLU. REDsec also deploys a discretized ReLU function, based on DoReFa-Net discretization [93]. 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. Moreover, 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 relatively simple. Bridging switches back to the binary domain, so that we can efficiently perform the remaining ReLU steps. The shift operation performs the discretization, and the output bits are determined by the model owner before training. The ReLU is performed by applying an `AND` of the inverse sign bit and the ciphertext output bits. Finally, in correspondence with DoReFa-Net [93], we clip the activation output. Here, an `OR` of the overflow bits is performed with the ciphertext output bits.

Novel Contributions. For sign activation, we adopt sign offsets; this is more related to TAPAS [78], but the authors never implemented it. For ReLU activations, REDsec is inspired by the procedure from DoReFa, and is adopted to the FHE domain. Bridging to the binary domain and performing a bitwise ReLU are unique features of REDsec.

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.

Sum Pooling. REDsec uses homomorphic-friendly SumPooling in place of AveragePooling, which is a standard practice in private neural network inference works [31] [35]. 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 [31], and the `OR` MaxPooling is often used for plaintext BNNs [17]. REDsec is the first to adopt 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, and 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 inefficient 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 novel approaches, namely *bidirectional bridging* and *lazy bootstrapping*. The first technique involves switching back and forth between the binary and integer domains for efficient operations. REDsec is the first work to enable this technique in TFHE, leveraging our data reuse optimization discussed in Section III-C1 (which avoids excessive bridging). REDsec's binary-to-integer switching allows us to use efficient integer additions, as opposed to previous binary domain works [52], [78] with excessively slow FHE runtimes. Our approach involves a single bootstrap, as well as *dividing the TFHE torus into multiple segments*. REDsec can revert back from the integer to the binary domain and take advantage of our optimized low-noise binary circuit operations discussed above (contrary to prior integer domain works [7]). Our 2-way bridging technique is discussed further in Section IV-C2.

REDsec also introduces *lazy bootstrapping*, which utilizes homomorphic noise estimations to limit the number of bootstraps and make it easier for the user to implement a neural network. It is known that additions and multiplications can affect the noise magnitude differently [30], 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 [23] and CKKS [11], but has not been sufficiently explored for 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. The TFHE torus can be divided into multiple segments (these can also be thought of as slices), and the bootstrapping operation can "fix" the FHE gate result to a specific region on that torus. In practice, only a certain number of gates can be evaluated on a ciphertext before bootstrapping is required to rescale the underlying plaintext value.

Our *noise auto-tuning* computes noise levels in an offline preprocessing step. In this case, noise checks are added after each operation and the noise auto-tuner designates the high-noise locations where bootstrapping is required as noise variance exceeds a certain threshold. This provides a significant speedup over gate bootstrapping, used in prior works such as SHE [52]. The threshold is a function of the plaintext space: the noise must not exceed the bounds of a torus slice (i.e., the region of the torus corresponding to a value modulo P), and the variance can therefore grow to $\frac{1}{2P}$, where P is the plaintext space or alternatively the number of torus slices. If the noise exceeds this limit, there is no guarantee that the underlying plaintext message still encodes the correct value (with high probability). 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 [12]. First, this parameter set corresponds to only 80 bits of security, while having a single parameter set offers low flexibility for arbitrary applications and algorithms. We introduce major changes to the library that offer customization of injected noise levels and alterations of the LWE polynomial degrees. The latter influences the sizes of both keys and ciphertexts. Such innovations allow users to find optimal parameter sets for their particular requirements.

Moreover, we introduce new 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 strategy 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 have a noise level greater than or equal to the highest noise level in the operands. For neural network inference, REDsec converts weights and biases into noiseless constants to interface with the user 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 (e.g., one can use Google Colab for debugging or rapid prototyping). After the proper training and validation dataset files are uploaded and linked, REDsec is ready to train the network. As soon as 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 mentioned 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 which are run on the cloud to perform classification in FHE.

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 programs are run on the client-side, none of which are computationally demanding. The first is a key generation process that allows users to specify a security level (in bits) and creates a homomorphic keypair. The user must also upload the evaluation key directly to the cloud server for the FHE inference procedure. Next, the user can utilize the second client module to prepare any inputs for secure inference: the raw bytes of the image are read, and checks are performed to ensure that the image is compliant with the network architecture. 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). The user uploads these ciphertexts to the cloud server to start the outsourced secure inference.

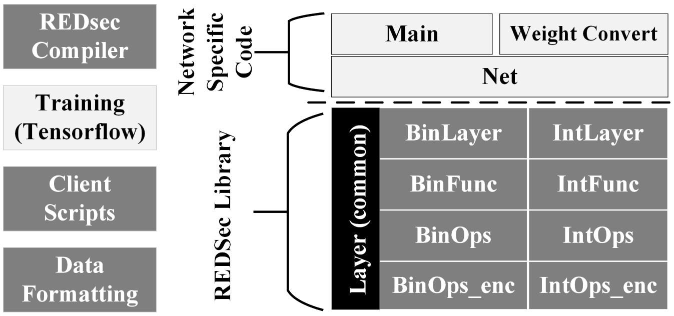


Fig. 2. **REDsec C++/Cuda Library:** Overview of different REDsec modules (described in Section IV-C1).

Lastly, when the cloud produces the encrypted results (which encode the scores for each class), the user can use the third client module for decryption. The class with the highest score indicates the most likely match with the input image and can be computed by taking 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 abstraction contains optimized implementations of convolution, fully connected, pooling, batch normalization, and quantize activation functions. OpenMP-based parallelization is also incorporated at this level.
- **Ops.** The Ops layer contains basic, low-level logic and arithmetic circuits that directly invoke the underlying cryptographic library. This level of abstraction offers 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 observe that bitwise-adder circuits form 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. 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 mentioned 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 cost of a single ciphertext addition. As long as the number of torus regions is sufficiently large to represent the range of desired plaintext values and the ciphertext noise is sufficiently low, there will be no rounding errors.

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.

Our multiplication procedure also enables the concept of *data reuse*, allowing ciphertext inputs to only be "multiplied" twice for the $\{-1, +1\}$ weights. As a result, we only need to bridge these two results, or twice per input, instead of bridging after every multiplication. Notably, *without our data reuse innovation*, bridging costs would be on the order of $85\times$ slower. Therefore, our data reuse approach enables us to run bidirectional bridging, and take advantage of the integer adder construction described above.

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 only need to make a copy of the ciphertext representing the MSB of the ciphertext vector, assuming the current encrypted value is represented 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 arithmetic circuits are 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.

Updates in (RED)cuFHE. Both the TFHE and cuFHE libraries only exposed 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. Therefore, (RED)cuFHE constructs leveled operations to fit with our lazy bootstrapping paradigm and offers redesigned ciphertext structures that can track the current noise variance. We remark that this variable accumulates differently for various operations involving encrypted data. In REDsec, this enables the proposed *auto-tuning* feature to predict the best places in the network to insert bootstrapping operations. Moreover, we parameterize (RED)cuFHE to allow for different configurations corresponding to various security levels, as determined using the LWE estimator framework [2].

Notably, previous work in cuFHE only supports HE operations on a single GPU, which limits the amount of potential parallelism. Conversely, (RED)cuFHE supports an arbitrary number of GPUs regardless of whether or not shared memory is available. Furthermore, 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. In (RED)cuFHE, 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.

V. Experimental Evaluation

To verify and test the efficiency of our framework, we conduct experiments using several network architectures to classify images from three datasets at various levels of complexity. We employ a g4dn.metal AWS instance for GPU experiments, hosting 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 offers 96 vCPUs running at 3.1 GHz. We configure both (RED)cuFHE and TFHE for 128 bits of security; specifically, we employ a ring dimension of 1024, an LWE dimension of 630, and an error distribution with a standard deviation of 2^{-15}. 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.

A. Impact of Optimizations

The REDsec framework incorporates many new innovations for binary and ternary neural networks. Specifically, the use of NOT gates for multiplications, data re-use, and bidirectional bridging are contributions whose speedup can be determined directly. All experiments in Figs. 3 and 4 use an r5.24xlarge AWS instance and the REDsec framework with modified convolution functions. Results were obtained running a four-hidden layer neural network. *Fully connected* comprises 1024–1024–512–512 output neurons, and *convolution* comprises 128–128–256–256 output neurons with a 3 × 3 convolution window on a 32 × 32 image using same padding.

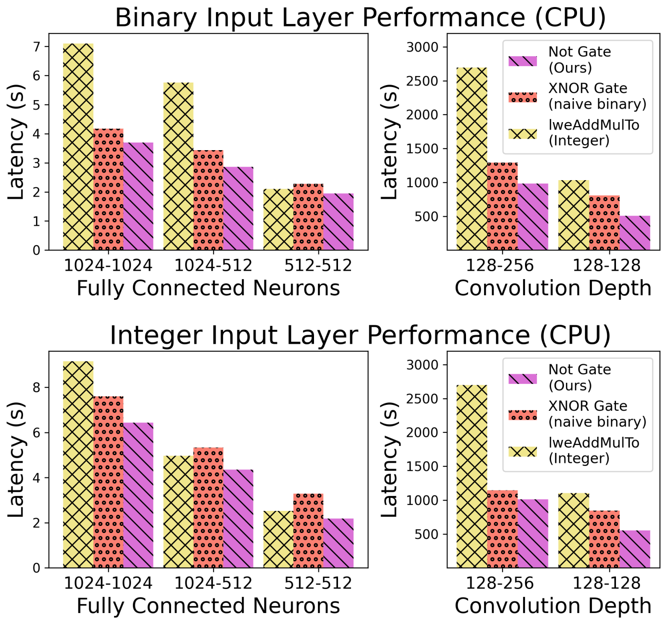


Fig. 3. **Layer Optimizations:** We compare REDsec to existing implementations using a fully connected layer of various input-output sizes. The XNOR and lweAddMulTo use the REDsec framework with modified convolution functions. The XNOR implementation includes bridging for integer additions, but performs multiplications with the standard XNOR. The integer domain comparison assumes no bridging, and uses `lweAddMulTo` (as used by FHE-DiNN [7]). The integer input implementation assumes a 10-bit integer input.

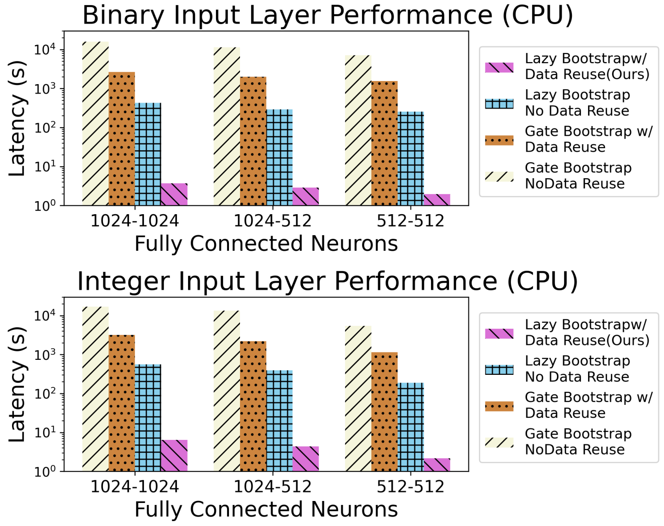


Fig. 4. **Layer Optimizations:** We compare with the standard gate bootstrapping methodology, where binary addition is performed with boolean gates and each gate is bootstrapped. We also report the impact of our data reuse methodology to emphasize the benefits of our ML optimizations. Data reuse eliminates the need for a bootstrap after multiplication, which in turn allows for bridging without a high-performance impact. Our results in magenta are identical to those shown in Fig. 3.

Bidirectional Bridging. Our bidirectional bridging allows REDsec to take advantage of *both binary and ternary opera-*

tions. One branch of prior FHE works operate in the binary domain [52], [78], with runtimes for MNIST networks in the order of hours due to inefficient adder circuits. Other FHE works operate exclusively in the integer domain [7], and cannot take advantage of any binary optimizations. Notably, REDsec's bidirectional bridging innovation enables our framework to benefit from both domains. A comparison between REDsec and integer domain calculations is shown in Fig. 3 for fully connected and convolution layers.

NOT-gate. Prior BNN works typically use the XNOR gate for multiplication, requiring a bootstrap for TFHE. Instead, REDsec uses the NOT gate and achieve considerable speedups for both fully connected and convolution layers for integer and binary inputs (Fig. 3). This is more pronounced with integer inputs, since the multiplication is more complex; with binary inputs, the observed cost of convolution is mostly attributed to bridging and integer addition.

Data Reuse. Comparing data re-use experimentally is a challenge since this is needed for efficient bridging. If no data re-use is applied, then we would need to bridge after every multiplication. Our experimental results for a variety of layer configurations (Fig. 4) show that data reuse for efficient bridging reduces latency by up to two orders of magnitude. Reusing data to enable efficient bridging is a key contribution in REDsec.

Lazy Bootstrapping. The lazy bootstrap measures noise growth during the first run of the neural network to determine the optimal bootstrapping locations. The alternative in TFHE is gate bootstrapping, where a bootstrap is performed after every logic operation. This aligns closely with the methodology employed by SHE [52]. Our lazy bootstrapping procedure also cuts down computation time by two orders of magnitude, as shown in Fig. 4. Overall, Fig. 4 demonstrates that without our data reuse and gate bootstrapping optimizations, the runtime of our fully connected layers is three orders of magnitude slower.

B. Fully Homomorphic Inference Results

The REDsec experimental results for various neural network architectures and network characteristics are shown in Table II. We preprocess all networks using a subtraction operation 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. These setup costs are not included in our inference measurements. However, we include the time taken for the user to encrypt an image and decrypt the classification results and find that this takes less than 3 seconds on a single CPU core for all configurations.

MNIST. For our experiments that evaluate the MNIST digit dataset, we use networks called Sign_{1024x1}, Sign_{1024x2}, and Sign_{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} 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 Sign_{1024x3} network, we achieve 98.0% accuracy. The ReLU networks have 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 presented in Figs. 5-6.

TABLE II. **SECURE INFERENCE EXPERIMENTS:** THE FIVE NETWORKS ARE EVALUATED USING FHE WITH 128-BITS OF SECURITY.

	Sign_{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
CPU-Eval (s)	12.3	18.4	1081	4622	7472.2
GPU-Eval (s)	3.6	8.2	229	1769.4	5918.5
Encrypt+Decrypt[2] (s)	1.4	1.4	1.4	1.4	2.6
Communication (MB)	1.9	1.9	7.5	7.5	367.5
Accuracy (%)	98.0	99.0	81.9	88.5	61.5 [1]

[1] Top-5 accuracy, as taken from pre-trained BYON benchmark [45].
[2] Combined client encryption/decryption on an Intel i7-8650U CPU.

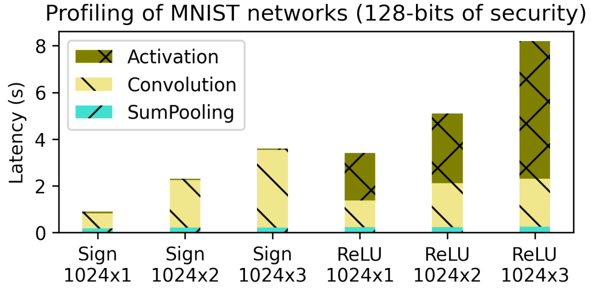


Fig. 5. **Performance Profiling:** The timing breakdowns for each MNIST network is shown. ReLUs are more expensive than sign, but are found to improve accuracy. Binary convolutions are more memory-sensitive than integer convolutions on the GPU, resulting in higher latency for this layer.

In addition to these base networks, we also ran smaller networks at 80-bits of security on a CPU host. In particular, using an AWS r5.24xlarge instance, REDsec's ReLU_{184x1} networks was evaluated in 2.2896s with 97.2% accuracy.

CIFAR. We use two architectures based on a CIFAR-10 BNN design from BinaryNet [17]. 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 [17]. 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 an open-sourced pretrained version of BinaryAlexNet for our evaluation [45]. 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.

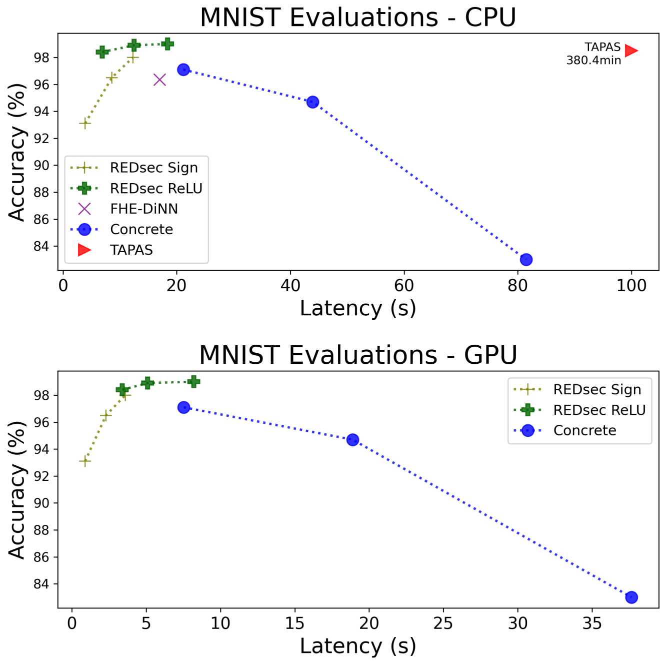


Fig. 6. **MNIST Performance:** We compare accuracy vs. latency for various works and neural network types. REDsec networks achieve higher accuracy with lower latency than earlier works, while REDsec does not suffer from rounding errors from programmable bootstrapping methods as in Concrete [13]. In addition, 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. All experiments are configured for 128 bits of security (except TAPAS that only supports 80 bits). REDsec uses ternary weight MNIST networks discussed in Section V-B.

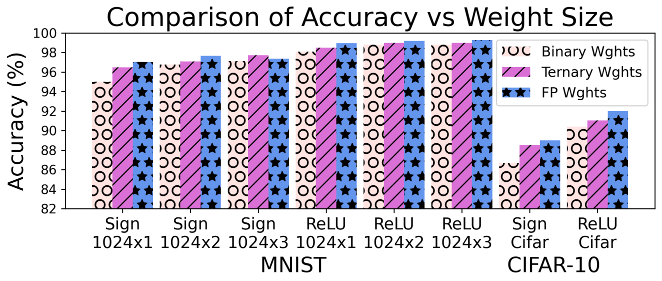


Fig. 7. **Experimental Comparison of Weight Size:** We report the accuracy for our MNIST and CIFAR-10 networks for different weight sizes. All networks use the same training scripts, only differing the weight size. The Larq library provides only limited support to full precision weights with sign activation. Details about BNN accuracy are provided in Section II-B.

For this network, we found that despite the memory transfer costs just outlined for GPU evaluation, REDsec can achieve homomorphic inference speeds of about 1.64 hours for the BinaryAlexNet with 128 bits of security (Table II). Overall, REDsec's performance benefits span a diverse set of network architectures with varying depths and complexity.

C. *Accuracy Comparisons*

Our Figure 7 presents accuracy comparisons for different weight sizes. Using ReLU activations helps prevent loss of information compared to sign, allowing us to achieve similar accuracy to floating point weight networks. We further found the latency for REDsec's ternary weight networks to be similar compared to the binary weight counterparts, with ternary achieving a 1-2% speedup on average. Our discussion about BNN accuracy and applications is found in Section II-B.

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 for FHE solutions are shown in Figure 6 and Table III.

A. *Multi-Party Computation Solutions*

The first class of secure inference solutions involves multi-party computation [4], [39], [41], [43], [51], [57], [69], [72], [83], [86]. These schemes typically 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 is incompatible with "fire-and-forget" style MLaaS applications. It also incurs communication overheads of hundreds of megabytes to gigabytes [39], [72]. Recent works such as Falcon [86] and CryptGPU [83] decrease this overhead. Also, several models use multiple servers and assume *non-collusion*, which may not always be guaranteed [43], [86]. REDsec supports non-linear activation functions while allowing for completely outsourced computation on a single server. Likewise, REDsec also supports the sign activation function, which is a popular choice for binary neural networks and can be evaluated very efficiently. Finally, some MPC works require clients to hold garbled circuits locally, resulting in large amounts of memory storage, which is typically infeasible for embedded edge and IoT devices [44]. Two MPC solutions that use binary neural networks are XONN and Samragh et al. [72], [77]. 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 costs and the requirement of involving the user in the calculation are still present. This is in addition to the other challenges of MPC methods already discussed.

B. *Leveled Homomorphic Solutions*

The second class of secure inference solutions is the pure LHE solution, including the works of CryptoNets [31], Faster CryptoNets [15], n-Graph HE [5] and CryptoDL [35]. These schemes use polynomial activations, including the square function x^2 [31], trained quadratic functions $a \cdot x^2 + b \cdot x$ [5] and n-degree polynomials for ReLU, tanh and sigmoid approximations [35]. These approximations either suffer from accuracy loss or generate a lot of noise as more terms must be evaluated for higher precision [60]. 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 throughputs, the latency is also high. Notably, these schemes do not support bootstrapping, which effectively limits applicability for deep FHE circuits. This is a key reason why batching does not play a large role in FHE evaluation: 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 upload thousands of images to be classified at once. On the other hand, REDsec is optimized for low latency through its use of the TFHE library.

TABLE III. **NETWORK MODEL COMPARISON:** MAX MACS CORRESPONDS TO THE LARGEST EXPERIMENTAL MODEL IN MILLIONS OF MULTIPLY-ACCUMULATE (MAC) OPERATIONS. REDSEC SCALES TO NETWORKS 10× LARGER THAN THE CONCRETE AND SHE FRAMEWORKS, AND ORDERS OF MAGNITUDE LARGER THAN FHE-DINN.

Performance	REDsec (this work)	Concrete [13]	FHE-DiNN [7]	TAPAS [78]	SHE [52]
Biggest Dataset[1]	ImageNet	MNIST	MNIST	MNIST	ImageNet[2]
Solution	FHE	FHE	FHE	FHE	LHE
Max MACs (M)	841.3	∼1.0	0.02	1.9	28
Weight Size	Micro	Small	Small	Small	Small
Training Lib.	TensorFlow	N/A	Custom	Custom	N/A
Input Type	Int	Int	Bin	Int	Int
Weight Type	Tern	Int	Int	Bin	Scaled
Activ. Type	Bin/Int	Int	Bin	Bin	Int
MNIST Train Time[4]	5 minutes	N/A[3]	11 hours	4.5 hours	N/A[3]

[1] Corresponds to the largest dataset that has actually been implemented.
[2] SHE uses ShuffleNet [90], which is an ImageNet architecture one order of magnitude smaller than AlexNet [42].
[3] No known open source implementation of neural networks.
[4] Based on a ReLU$_{1024x1}$ network.

Finally, LHE schemes are not scalable to deeper networks due to noise. One can use bootstrapping to make these schemes (e.g., BGV and CKKS) FHE, 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 [48]. Conversely, our approach uses TFHE-based schemes that offer efficient bootstrapping methods and allow for BYON architectures with unlimited depth.

One LHE BNN work [66] 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 the full-precision layers to the edge device company server. The user gives this server private data, which is run in the clear through part of the network. The result is transmitted to the cloud server, which uses HE to run the BNN layers and the model is protected since the edge server can't access the entire model. However, it assumes that the server can be trusted with sensitive data and requires the edge device company to maintain an expensive server. REDsec 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 [66].

Another leveled homomorphic work is SHE (by Lou and Jiang [52]), which differs from previous networks in that it uses a leveled variant of TFHE. SHE also offers partial support for FHE, but these results are extrapolated and practically infeasible. Lou and Jiang also discuss the implementation of a full-precision neural network by approximating weights to a power of two, and this multiplication optimization allows the circuit to be an XNOR followed by a bitshift [52]. Although mixed-precision BNNs sometimes use this technique, it is not yet supported in the current version of the Larq library [27], [50], [52], [80], [93]. The main limitation of SHE is that it builds its architecture upon the Boolean logic gate operation in TFHE, which is inherently very slow. This increases inference times compared to using integer arithmetic [14], [52]. Since bootstrapping is not utilized in the leveled mode of operation in SHE, the network still suffers from the scalability problems discussed for other LHE frameworks. For FHE mode, Lou and Jiang extrapolate that their framework in gate bootstrapping mode would take over 3 hours to run MNIST inference (our work performs inference in 2.3 seconds for the same 80-bit security level). When evaluating on the same r5.24xlarge AWS instance as REDsec, SHE took 627 seconds for a single 1024-length dot product; REDsec computes thousands of these dot products in our MNIST networks in just a few seconds. While SHE has no working AlexNex implementation, its authors extrapolate that a leveled implementation would take 24.7 hours to run [52], compared to our experimental evaluation of AlexNet that runs in 1.64 hours. Finally, the available implementation of SHE covers only basic functional units instead of a complete secure inference neural network [53].

C. Fully Homomorphic Solutions

The third class are FHE solutions, including REDsec. Compared to other FHE works, and all other works, 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 full circles in the Table). Few existing works use the FHE approach, but since the introduction of TFHE with its efficient bootstrapping procedure, several works have proposed related solutions, as summarized in Tables III and IV.

DOReN [55] improves on SHE with an updated Boolean adder design. However, as discussed, adders are inefficient due to the large circuits required to evaluate them. REDsec's bridging implementation is far more efficient than gate bootstrapping implementations, as shown in Figs. 3 and 4. DOReN also uses the high-throughput, high-latency BGV library, which has significantly slower bootstrapping times than TFHE. As a result, DOReN's smallest CIFAR-10 network has a 9 hour latency with 73.9% accuracy, which is significantly slower and less accurate than REDsec (Table II). FHE-DiNN [7] 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. Fig. 6 compares to FHE-DiNN's 30-neuron single hidden layer network using an r5.24xlarge instance. Finally, FHE-DiNN is harder to train, not configurable, and relies on a custom Keras implementation instead of an optimized library for BNN training [6], [7]. 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 [13]. Their sample neural network codebase is not published, making it difficult to compare to our work [89]. Nevertheless, based on the description, their biggest network has around one million multiply-accumulate (MAC) operations (92 inputs·92 outputs·100 layers) [13]. 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

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

Features	FHE	Dataset Size[3]	≤ 2 Parties[4]	Conv-olution	Fully Connected	MaxPool	SumPool	Batch Norm	Sign Activ.	ReLU	Int. Inputs	Configurable[5]	Implemented [6]	Comm. Overhead[7]	Client Storage
Cheetah [69]	○	●	●	●	●	◐[1]	○	○	○	◐[1]	●	○	●	GB	GB
Gazelle [39]	○	◐	●	●	●	◐[1]	○	○	○	◐[1]	●	○	●	GB	GB
SecureML [57]	○	○	●	○	●	○	○	○	○	◐[1]	●	○	●	GB	GB
XONN [72]	○	◐	●	◐[1]	◐[1]	◐[1]	○	●	◐[1]	○	●	○	●	GB	GB
MP2ML [4]	○	○	●	●	●	◐[1]	●	○	○	◐[1]	●	○	●	GB	GB
MiniONN [51]	○	◐	●	◐[1]	◐[1]	◐[1]	◐[1]	●	○	◐[1]	●	○	●	GB	GB
Falcon [86]	○	◐	●	●	●	◐[1]	◐[1]	●	◐[1]	◐[1]	●	○	●	MB/GB	GB
CryptFlow [43]	○	●	●	●	●	◐[1]	●	○	○	◐[1]	●	●	●	GB	GB
CryptGPU [83]	○	●	○	◐[1]	◐[1]	○	◐[1]	●	◐[1]	◐[1]	●	●	●	MB/GB	GB
CryptTen [41]	○	●	○	◐[1]	◐[1]	◐[1]	◐[1]	●	◐[1]	◐[1]	●	●	●	GB	GB
CryptoNets [31]	○	○	●	●	●	○	●	○	○	○	●	○	●	GB	MB
Faster CryptoNets [15]	○	○	●	●	●	○	●	●	○	◐[2]	●	○	●	GB	MB
n-GraphHE [5]	○	●	●	●	●	○	●	●	○	●	●	●	●	GB	MB
CryptoDL [35]	○	◐	●	●	●	○	●	●	○	◐[2]	●	●	●	GB	MB
Concrete [13]	●	○	●	●	●	●	●	●	○	●	●	○	●	MB	MB
FHE-Dinn [7]	●	○	●	○	●	○	○	○	●	○	○	○	●	MB	MB
Hopfield [38]	●	○	●	○	○	●	○	○	●	○	○	○	●	MB	MB
SHE [52]	●	◐	●	●	●	●	○	●	○	●	●	○	○	MB	MB
DOReN [55]	●	◐	●	●	●	●	○	●	○	●	●	○	●	MB	MB
Tapas [78]	●	○	●	●	●	○	○	○	●	○	○	○	○	MB	MB
REDsec	●	●	●	●	●	●	●	●	●	●	●	●	●	MB	MB

[1] Requires more than one computing parties.

[2] Operations are polynomial approximations.

[3] Dataset size on scale of ○ MNIST ◐ CIFAR10 ● Imagenet.

[4] A third computing party is not required.

[5] Open sourced framework with bring-your-own-network.

[6] Results from actual implementation, not extrapolated.

[7] Communication overhead of AlexNet-sized networks.

Concrete, and our convolution requires one bootstrap for each input, where Concrete requires two bootstraps for every multiplication operation [13]. Further, programmable bootstrapping yields low precision with efficient parameters, which results in rounding errors; this is evident in the fact that deeper networks with more layers result in less accuracy. Concrete's NN-20 network for MNIST classification offers 97.1% accuracy while the NN-100 (5× more layers) only yields 83.0% classification accuracy [13]. Fig. 6 shows this accuracy degradation, and REDsec's ability to achieve higher accuracy for lower latency.

TAPAS presents theoretical approaches to secure inference [78]. A 1024x2 MNIST network on TAPAS on the same 96-core AWS instance used for REDsec, ran in 380.5 minutes with an accuracy of 98.5%. This is more than three orders of magnitude slower than REDsec for comparable accuracy and lower security (Fig. 6). In addition, they did not explore implementation details such as ternary networks, data reuse, and integer activations. Finally, since there is no material neural network, it remains difficult for future works to expand on their approach. A Hopfield network architecture was recently proposed for secure inference [38], which is a binary recurrent neural network of one or a few layers. It was first proposed as an early machine learning networks, and some recent applications have made use of them because neurons can run independently in parallel [36], [81]. Given its binary nature, this network architecture is a good choice for FHE inference, but it is not as widely used as conventional BNNs for most applications [38], [80]. 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 [7], [13], [52] assume an honest user and an honest-but-curious cloud which evaluates the function correctly but attempts to glean information about user inputs. In future work, we will expand the current FHE threat model to include malicious users; specifically, we will investigate adversarial machine learning attacks on the model's intellectual property, which can be performed on any MLaaS model but requires a high number of queries and can only produce an accuracy close to the original model [16], [64], [84], [88]. This is an important research direction as privacy-preserving ML solutions often avoid implementing a max function for outputs, which can make model inversion attacks theoretically possible. Fortunately, binary neural networks are less prone to adversarial attacks [26], [85]. In future work, we plan on investigating efficient FHE defenses against these threat models. Further, we will incorporate residual networks (like ResNet) in REDsec to allow for different styles of DNNs and determine the feasibility of FHE for these networks.

VIII. CONCLUSION

In this work, we present REDsec, an end-to-end framework for binary and ternary neural network training and secure inference using fully homomorphic encryption. To enable BYON support, we incorporate a compiler to output both TensorFlow training and secure inference code for easy adoption by data scientists and researchers. Our 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 introduce the (RED)cuFHE library for state-of-the-art GPU acceleration of TFHE, with support for multiple GPUs, GPU data transfer coordination, encryption of constants, leveled gate operations, data reuse and bridging between binary and integer ciphertexts to optimize FHE operation performance. We demonstrate the benefits of our framework by evaluating MNIST, CIFAR-10, and ImageNet, and report significant performance improvements compared to related works.

Resources

The REDsec framework and the (RED)cuFHE library are available online as open-source software under the MIT license [25], [32].

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] ——, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] W. Dai and B. Sunar, "cuFHE (v1.0)," https://github.com/vernamlab/cuFHE, 2018.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

[24] 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.

[25] L. W. Folkerts, C. Gouert, and N. G. Tsoutsos, "REDsec: Running Encrypted Discretized Neural Networks in Seconds," https://github.com/TrustworthyComputing/REDsec, 2023.

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

[27] 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

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] C. Gouert and N. G. Tsoutsos, "(RED)cuFHE: Evolution of FHE acceleration for Multi-GPUs," https://github.com/TrustworthyComputing/REDcuFHE, 2022.

[33] 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.

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

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

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

[41] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[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] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 336–353.

[44] A. Kumbhare, Y. Simmhan, and V. Prasanna, "Cryptonite: A secure and performant data repository on public clouds," in *2012 IEEE Fifth International Conference on Cloud Computing*. IEEE, 2012, pp. 510–517.

[45] Larq, "BinaryAlexNet on Larq Zoo Pretrained Models," https://docs.larq.dev/zoo/api/literature/#binaryalexnet.

[46] ——, "Building BNNs," https://docs.larq.dev/larq/guides/bnn-architecture/.

[47] ——, "Training BNNs," https://docs.larq.dev/larq/guides/bnn-optimization/.

[48] 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.

[49] F. Li, B. Zhang, and B. Liu, "Ternary weight networks," *arXiv preprint arXiv:1605.04711*, 2016.

[50] 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.

[51] 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.

[52] 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.

[53] ——, "SHE: A Fast and Accurate Deep Neural Network for Encrypted Data," https://github.com/safednn/SHE, 2019.

[54] 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.

[55] S. Meftah, B. H. M. Tan, C. F. Mun, K. M. M. Aung, B. Veeravalli, and V. Chandrasekhar, "Doren: toward efficient deep convolutional neural networks with fully homomorphic encryption," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3740–3752, 2021.

[56] 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/

[57] 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.

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

[59] NuCypher, "nuFHE (v0.0.3)," https://github.com/nucypher/nufhe, 2019.

[60] S. Obla, X. Gong, A. Aloufi, P. Hu, and D. Takabi, "Effective activation functions for homomorphic evaluation of deep neural networks," *IEEE Access*, vol. 8, pp. 153 098–153 112, 2020.

[61] 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.

[62] P. Panda, "Quanos: adversarial noise sensitivity driven hybrid quantization of neural networks," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 187–192.

[63] 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.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] 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.

[69] B. Reagen, W.-S. Choi, Y. Ko, V. T. Lee, H.-H. S. Lee, G.-Y. Wei, and D. Brooks, "Cheetah: Optimizing and accelerating homomorphic encryption for private inference," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 26–39.

[70] 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.

[71] ——, "The learning with errors problem," *Invited survey in CCC*, vol. 7, no. 30, p. 11, 2010.

[72] 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*, 2019, pp. 1501–1518.

[73] 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.

[74] 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.

[75] 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.

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

[77] 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.

[78] 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.

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

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

[81] 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.

[82] 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.

[83] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1021–1038.

[84] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing Machine Learning Models via Prediction APIs," in *25th USENIX security symposium*, 2016, pp. 601–618.

[85] 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.

[86] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, "Falcon: Honest-majority maliciously secure framework for private deep learning," *arXiv preprint arXiv:2004.02229*, 2020.

[87] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science*. IEEE, 1986, pp. 162–167.

[88] 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.

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

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

Appendix A: DiNN Foundations in REDsec

In this appendix, we present the mathematical foundations of our discretized neural networks. Basic concepts are adopted from generic binary neural networks, and 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 8. 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. [20]. BatchNorm is placed before the activation function, and take the form of

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

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 converted into the form

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

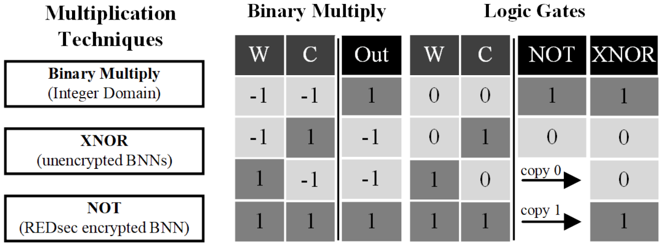
$$m = \frac{\gamma}{\sqrt{\sigma^2+\eta}} \tag{10}$$

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

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 [20].

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


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 →	0
1	1	1	1	1	copy 1 →	1

(a) Strategies (b) Truth Tables

Fig. 8. **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.

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

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

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 $+1 \mapsto +2$. 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.

Appendix B: GPU Modules for Encrypted Computation

GPUs speed up homomorphic operations significantly over strictly CPU-based systems. For example, a GPU can achieve over 37× speedup compared to a CPU for bootstrapping operations using the TFHE scheme [59]. 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), our REDsec library provides GPU support for all homomorphic operations through the use of CUDA code, based on the major (RED)cuFHE optimizations over prior works. Notably, using 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.

In order to maximize resource utilization for any number of GPUs, we leverage our 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.