SecONNds: Secure Outsourced Neural Network Inference on ImageNet in a second

***Abstract*—SecONNds is a non-intrusive secure inference framework specifically optimized for outsourced image classification on ImageNet, achieving groundbreaking subsecond performance. By integrating a novel fully Boolean GMW protocol for secure comparison (Yao's millionaires' problem) with preprocessed Beaver's bit triples generated from silent random oblivous transfer, SecONNds attains a 20-*fold online speedup* in nonlinear operations compared to existing solutions while requiring a lower communication. This efficiency is further enhanced with Number Theoretic Transform (NTT) preprocessing and GPU acceleration for Homomorphic Encryption operations, resulting in speedups of $1.4\times$ on CPU and $11\times$ on GPU for linear operations. Furthermore, SecONNds introduces a bit-exact variant, SecONNds–P, which ensures full-precision-verifiable results in secure computation compared to plaintext. We evaluated SecONNds on a 32-bit quantized SqueezeNet model and achieved an end-to-end inference time of just 0.98 seconds on GPU and 4.05 seconds on CPU, with a total communication of 382 MiB. SecONNds sets a new standard for privacy-preserving neural network inference, combining exceptional performance with robust security and reduced computational load for users, making it ideal for privacy-sensitive applications on resource-constrained environments. The entire SecONNds framework is completely open source, to promote the use, development, and transparency of secure AI technologies.**

I. Introduction

Machine learning (ML) has become ubiquitous, with pre-trained neural networks (NNs) playing a pivotal role in numerous applications that shape our daily interactions. Many of these applications are outsourced and involve processing user data on remote cloud servers, which presents significant privacy risks to users. In response, the security and privacy research community has introduced secure inference frameworks. These solutions allow users to utilize pre-trained NNs for inference without exposing their raw input data or the confidential model parameters of the NN. Using cryptographic techniques, these frameworks ensure the protection of all private data from all parties involved, including the user's input and output of private inference, and the parameters of the service provider's proprietary model.

Neural networks are very diverse in architecture with respect to the task at hand, but essentially they are composed of an alternating series of multidimensional linear operations, and nonlinear operations that are mostly unary. Convolutional Neural Networks (CNN) are one of the most popular classes of architectures that have become essential for computer vision tasks. CNNs feature multidimensional convolution operations to match learned spatial features with the input image, ReLU operations to pick the task related matches, and pooling operations to downsample the matches. CNNs kicked off the Deep Learning era with models growing larger and larger in terms of parameters. These large models have very high computing requirements and are impossible to run on resource-constrained devices such as mobile and embedded systems. To effectively implement secure inference while preserving functionality and performance, it is very important to codesign computation algorithms for low-level neural network operations with cryptographic primitives that provide privacy in light of the resource allocation of both parties.

Employing a zero-trust model for secure inference means the framework not reveal any private data, and should not involve any entity other than the user and the service provider, demanding a purely 2-party Security Model. This necessitates cryptographic primitives for secure 2-party computation protocols (2PC): Garbled Circuits (GC) [1], Goldreich-Micali-Wigderson (GMW) [2], Homomorphic Encryption (HE)[3]. GC is a one-round protocol for securely evaluating Boolean circuits and was the first solution proposed for Secure Computation. GMW enables the secure evaluation of boolean and arithmetic circuits on fixed-point data with Oblivious Transfer (OT) [4] and Linear Secret Sharing Schemes (LSSS) [5], but it is an interactive protocol that requires constant communication between parties for every operation. Both GC and GMW assume symmetric resource allocation for both parties and necessitate an equal amount of local computation on both ends.

HE is an encryption scheme that preserves a structure in the encrypted data allowing one to perform computations on it that reflect as simple operations on the secrets inside encryptions. While partial HE schemes[6], [7] offer a very small set of operations for computation, fully HE schemes [3], [8], [9] enable Boolean and arithmetic circuits but are very slow. Modern Leveled-HE schemes [10], [11], [12], [13] strike a good balance by allowing arithmetic circuits on fixed-point data up to a fixed size with practical performance. The main advantage of HE is in securely outsourcing the computation; the user sends encryptions to the server that performs all computation and returns the encrypted input to the user. This makes HE best suited to asymmetric scenarios involving users with resource-constrained devices and a service provider with a powerful cloud server.

Network and Distributed System Security (NDSS) Symposium 2024
26 February - 1 March 2024, San Diego, CA, USA
ISBN 1-891562-93-2
https://dx.doi.org/10.14722/ndss.2024.23xxx
www.ndss-symposium.org

Since secure computation protocols work with fixed point, we must first convert neural networks operations to fixed point representations. Thankfully, this problem has been thoroughly studied in machine learning in terms of quantized neural networks (QNN) [14]. The primary effort in evaluating QNNs comes from linear operations; in fact, almost all of the parameters of a model are used only to compute specific linear combinations of the inputs to their respective layers, and hence a model's size directly corresponds to the number of linear operations involved. With GC or GMW the communication footprint of a linear operation will scale with the product of the flattened sizes of the operands. With HE the communication footprint scales only with the sum of the sizes of the input and output.

Today, leveled-HE outperforms any other 2PC primitive in evaluating high-dimensional linear algebra operations on encrypted data [15]. On the other hand, nonlinear operations are impossible with leveled HE and are indispensable in Neural Networks. Nonlinear operations involve comparisons, and in fact this was one of the first problems studied in secure computation by Andrew Yao [1], who dubbed it the Millionaires' Problem $\mathsf{MILL} = \mathbb{1}\{i_0 > i_1\}$, which indicates if party-0's input is greater than party-1's input. The bulk of the online runtimes of nonlinear operations in state-of-the-art secure inference protocols come from the Millionaires' Protocol (secure comparisons). To compare b bit secrets, GC involves a ciphertext expansion of $O(\lambda b)$.[1] When employing a GMW protocol initialized with Silent OT Extension [16] the same task involves an effective overhead of only $O(b)$, greatly propelling its performance over GCs.

Contributions: We present SecONNds, a new framework for secure inference that includes:

- A new solution to the Millionaires' Problem, $\mathcal{F}_{\mathsf{MILL}}$, which features a fully Boolean GMW Protocol with Beaver's Bit Triples (triples) [17] generated using silent Random OT (ROT) [18], that achieves faster runtimes and lower communication than prior work. We augment our protocol with a buffer for triples to preprocess triples when offline (offline triples) and implement a chunked triple generator optimized for Silent OT.
- New 2PC Protocols for: ReLU, Max Pooling and Truncation with $\mathcal{F}_{\mathsf{MILL}}$ featuring offline triples; Linear algebra operations with the BFV HE scheme [11], [12] featuring one-time preprocessing with the Number Theoretic Transform (NTT) [15] and server-side GPU acceleration [19]. Our protocols achieve online speedups of $17\times$ for ReLU, $24\times$ for Max pooling, $2.3\times$ for truncation and $1.4\times$ for convolution with HE on GPU.
- A fully open-source implementation[2] of end-to-end (E2E) secure inference with ReLU, Max Pooling and Truncation using $\mathcal{F}_{\mathsf{MILL}}$ with offline triples along with the GPU accelerated protocols for linear operations. Our implementation acheives an E2E runtime of 0.98 secs for secure inference on ImageNet [20] with a 32 bit quantized SqueezeNet Model [21] involving 382 MiB of total communication. Our E2E performance is $9\times$ faster online compared to state-of-the-art.

[1] λ is the Computational Security Parameter, typically 128.

[2] Concealed, as per Artifact Evaluation (AE) process

II. Background

A. Mathematical Notation

In this section, we introduce the mathematical notation used throughout this paper. Integer vector fields are denoted by $\mathbb{Z}_N^m$ where m is the dimensionality of the vector space and N is the field modulus. Polynomial rings are denoted by $\mathcal{R}_Q^N = \mathbb{Z}_Q[X] \bmod \left(X^N + 1\right)$ for polynomials of degree less than n with coefficients from $\mathbb{Z}_Q$ where N is the polynomial degree modulus and Q is the coefficient modulus.

Scalars in $\mathbb{Z}_N$ are denoted by the normal text x or Y. Vectors in $\mathbb{Z}_N^m$ are denoted by bold lowercase letters $\mathbf{x}$, and matrices in $\mathbb{Z}_N^{p\times q}$ by bold uppercase letters $\mathbf{X}$. A polynomial in $\mathcal{R}_Q^N$ in denoted as $\overline{\mathrm{p}}$. A HE plaintext encoding a secret $\mathbf{m}$ is denoted as $\overline{\mathrm{pt}}_{\mathbf{m}}$ and its ciphertext is denoted as $\overline{\mathbf{ct}}_{\mathbf{m}}$.

The bitwise complement of a scalar x is denoted by x'. The operators $\oplus, \wedge$ are reserved for addition and multiplication in $\mathbb{Z}_2$. Party-p's linear secret share of i over $\mathbb{Z}_N$ is denoted by $\langle i \rangle_p^N$, therefore $i = \langle i \rangle_N^p + \langle i \rangle_N^p \bmod N$. The indicator function is denoted by $\mathbb{1}\{\text{condition}\}$, it is 1 if the condition is satisfied, and 0 otherwise.

B. Oblivious Transfer

Oblivious Transfer (OT), first introduced by Rabin [22], is a protocol that enables the exchange of secret messages between two parties. In a b bit n-choose-1 OT, denoted by $\binom{N}{1}$–OT_b, one party (sender) inputs a set of n b bit values $\{x_0, x_1, \cdots, x_{n-1}\}$, and the other party (receiver) inputs a choice $c \in [0, n)$ corresponding to an element in the set. The receiver learns only one of the sender's input, x_c corresponding to its selection c, oblivious to the sender who does not learn anything. Correlated OT (COT) and Random OT (ROT) are two simpler variants of OT that are powerful cryptographic primitives. In $\binom{2}{1}$–COT_b, the sender inputs a private b bit correlation δ and learns a random b bit value r while the receiver learns the b bit $r + c \cdot \delta$. In $\binom{n}{1}$–ROT_b, the sender does not input anything and learns n random b bit values $\{r_0, r_1, \cdots, r_{n-1}\}$ while the receiver learns only one of them, r_c for its choice c.

Initial constructions of OT utilized public-key cryptography which incurred large overheads in terms of both computation and communication. OT extension protocols, first conceptualized by Beaver [17] and later realized by Ishai et al. [23] (IKNP), reduce this overhead by generating a $O(n)$ OTs using lightweight symmetric key cryptographic operations from $O(n/k)$ public-key OTs (*base OT*) where k is a fixed constant parameterized by the OT extension protocol. Recently, Boyle et al. [16] devised a new OT extension protocol, namely *silent OT extension*, which significantly reduces the number of base OTs required to $O(\log n)$ at the expense of more local computation.

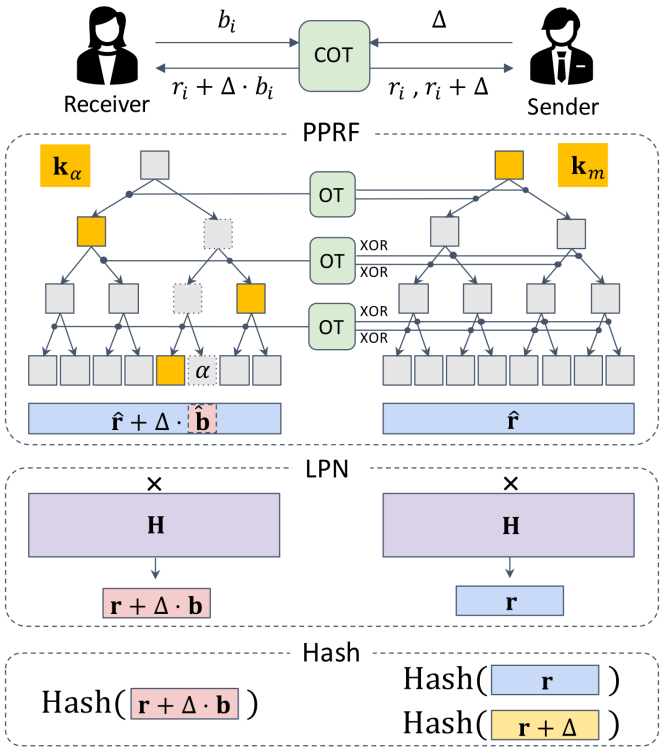

Fig. 1. High-level illustration of silent OT protocol.

C. Silent OT Extension

The core components of Silent OT Extension are Punctured Pseudo-Random Function (PPRF) and Encoding with Learning Parity with Noise (LPN). Figure 1 illustrates the high-level operations involved in silent OT. A Pseudo-Random Function (PRF) is a deterministic random function that maps random values from a small domain to pseudo-random numbers in a larger domain. A *punctured* pseudorandom function (PPRF) [24], [25] is a keyed PRF that can be evaluated on all points in the PRF's domain when used with a *master key* $\mathbf{k}_m$, and can be evaluated on all points except α when used with a *punctured key* $\mathbf{k}_\alpha$. The PRF is generally constructed as described by Goldreich, Goldwasser, and Micali (GGM tree) [26] which involves tree expansion of $1 + \log n$ levels with one Hash operation per node for n OTs.

The LPN assumption [27] (in dual form) states that $(H, r \cdot H) \overset{c}{\approx} (H, b)$ where $H \in \mathbb{Z}_2^{m \times n}$ is a code matrix with $m > n$, $r \in \mathbb{Z}_2^m$ is a random sparse vector, and $b \in \mathbb{Z}_2^n$ is a uniformly random vector. In Silent OT, LPN is used to encode the sparse outputs of GGM tree expansion into dense outputs. This involves computing a linear transformation with a large code matrix $H \in \mathbb{Z}_2^{2n \times n}$ for n OTs. The LPN encoding outputs correspond to $\binom{2}{1}$–COT_b, the sender computes two hashes of its output, while the receiver computes one hash to convert $\binom{2}{1}$–COT_b to $\binom{2}{1}$–ROT_b. To perform a $\binom{2}{1}$–OT_b, the sender uses its two outputs of $\binom{2}{1}$–ROT_b as one-time pads to mask its inputs of $\binom{2}{1}$–OT_b and sends it to the receiver who unmasks the corresponding message with its output of $\binom{2}{1}$–ROT_b.

Overall, while silent OT offers significant gains in communication, it involves more intensive computation compared to IKNP-style protocols. The primary bottleneck in the computation arises from LPN encoding which involves a complexity of $O(n^2)$ that is quadratic in the size of the required number of OT.

D. Secure Computation

Garbled Circuits perform the computation using Boolean circuits. One party, the garbler, assigns two random labels per wire (for values $0, 1$) and generates a garbled truth table per gate containing symmetric encryptions of the output label with the input labels as keys and sends the garbled tables and the labels of its inputs in the circuit to the other party, the evaluator. The evaluator securely learns the labels for its inputs in the circuit using OTs and evaluates the circuit with a series of decryptions for each garbled table. Although GC is a one-round approach, it introduces severe overheads in memory and computation, since each bit is represented with a large λ bit ciphertext and each bit-operation in the circuit requires expensive descriptions.

GMW is an interactive protocol in which, for each operation, both parties compute locally on secret shares of private values and obliviously share intermediate results using OT to ensure correct and secure computation. The secure XOR operation is straightforward; each party locally computes the XOR of their respective shares without any interaction. This is possible because XORing the shares naturally aligns with the properties of secret sharing in $\mathbb{Z}_2$, preserving both correctness and secrecy without additional communication. Conversely, the secure AND operation requires interaction; the parties typically use a multiplication triple which is generated using OT. They then exchange and combine specific bits of their shares and the multiplication triple to compute the output securely, ensuring that no information about the operands is revealed during the computation. GMW is more efficient in terms of memory since the secret shares are the same size as the original data. The computation is also simpler because it can be expressed as arithmetic circuits.

Both GC and GMW involve an amount of communication that increases in tandem with the total number of multiplication operations needed. While GC achieves this exchange in a constant number of communication rounds, GMW requires one round of communication for every multiplication. The round complexity of GMW can be significantly reduced with circuit randomization [28] and correlated pseudorandomness [17]. When it comes to computation, while both protocols offer the ability to perform any computation, the primary challenge with them is the nearly-equal computational effort demanded from both parties, and hence they do not offer the ability to outsource computation.

Homomorphic Encryption (HE) is an advanced technology that enables computations on encrypted data without needing decryption keys or access to the original plaintext. In an HE system, the client encrypts data and sends it to the server, which performs computations on the encrypted data

and returns the results to the client for decryption. A major benefit of HE is its ability to separate communication volume from computational workload. The data transfer is based solely on input and output sizes. Additionally, HE places most of the computational effort on the server, while the client handles encryption and decryption. Fully Homomorphic Encryption (FHE) schemes, introduced by Gentry [3], support arbitrary computations on encrypted data using a bootstrapping process for unlimited computation. Notable examples include TFHE [9] and FHEW [8], which focus on optimizing bootstrapping. While FHE reduces computation-dependent communication, it incurs significant computational overhead. Leveled HE schemes allow a fixed number of computations without bootstrapping, enhancing speed but increasing ciphertext size and communication needs. These schemes' computational capacity is determined by the multiplicative depth, the number of sequential multiplications, which is beneficial for tasks like large matrix multiplications where each multiplication has a multiplicative depth of 1.

Ring Learning With Errors (RLWE) [29] underpins the construction of most of the popular HE schemes like BGV [10], BFV [11], [12], and CKKS [13]. RLWE is a computationally hard problem that involves solving for a small error in polynomial equations. RLWE-based HE schemes operate on polynomial rings, $\mathcal{R}_Q^N$ where N is defined as the *polynomial modulus degree* and Q is defined as the *coefficient modulus*. For the BFV and BGV schemes, a plaintext $\overline{\text{pt}}$ is an element from $\mathcal{R}_P^N$, where P ($< Q$) is defined as the plaintext modulus. The CKKS scheme is designed for approximate computation works with different mechanics where plaintexts are elements in $\mathcal{R}_Q^N$ and it does not define a plaintext modulus. Encryption involves encoding a secret vector $\mathbf{m} \in \mathbb{Z}_P^N$ into a polynomial $\overline{\text{pt}}_{\mathbf{m}}$, and then computing $\overline{\mathbf{ct}}_{\mathbf{m}} = (\bar{\text{a}}, \bar{\text{b}})$ with a random polynomial $\bar{\text{a}} \in \mathcal{R}_Q^N$, $\bar{\text{b}} = \left(\bar{\text{a}} \cdot \overline{\text{sk}} + \delta \cdot \overline{\text{pt}}_{\mathbf{m}} + \epsilon \cdot \bar{\text{e}}\right)$ where $\overline{\text{sk}}$ is the secret key, δ & ϵ are scheme-defined constants, and most importantly $\bar{\text{e}}$ is a small error sampled from a discrete Gaussian distribution whose standard deviation is determined by the security parameter λ.

Operations on HE ciphertexts are implemented by performing certain computations on their respective polynomials. RLWE HE schemes offer vector addition, multiplication and rotation. Multiplications are optimized using transformations like the Number Theoretic Transform (NTT), which reduces polynomial multiplication complexity from $O(N^2)$ to $O(N \log N)$. However, rotations, requiring key-switching with special keys $\overline{\mathbf{ek}}$, are still computationally demanding due to the size of these keys and their number based ciphertexts, $\bar{\text{e}}$, central to the hardness of RLWE, increases with each computation on the ciphertexts. The total error tolerance is bounded by the ring's coefficient modulus Q, influencing both the security level and operation speed of the HE scheme. Operations like additions and rotations cause additive error growth, while multiplications cause multiplicative error growth, necessitating strategic operation placement and parameter selection to control error expansion and optimize performance.

E. Secure Inference

Secure inference frameworks utilize foundation protocols for performing core integer operations on encrypted data to build higher-level protocols to perform neural network operations on fixed-point data. *CryptoNets* [30] is the first secure inference framework to run a Convolution Neural Network on MNIST [31]. It is constructed entirely with HE and performs secure inference in 1 round. It uses arithmetized CNNs that are finetuned with matrix multiplications (fully connected layers) in place of convolutions and polynomial activations in place of nonlinear ones.

MiniONN [32] is the first non-intrusive[3], mixed-protocol framework to perform inference on CIFAR-10 [33]. It uses GC, GMW and HE to implement protocols for most of the popular CNN operations. It implements exact protocols for piecewise linear functions like ReLU and for smooth functions, its protocols approximate the functions with splines. MiniONN features an offline preprocessing phase to set up beaver's bit triples [17] using HE, only for multiplications in the linear layers. During the online phase, which involves evaluating the inference result of the private image, they use GMW with preprocessed triples for the linear layers and GC for the nonlinear layers. MiniONN's novel mixed protocol design requires communication after every layer introducing additional communication rounds and yet offers two orders of magnitude better performance than CryptoNets. All following secure inference frameworks strictly targeting E2E performance adhere to this Mixed-protocol design.

Gazelle [34] implements the first purpose-built algorithm for performing convolutions on 3-D data using HE. It combines these HE protocols for linear layers with a GC protocol for secure comparisons in $\mathbb{Z}_P$ where P is a HE plaintext prime modulus and showed an order of magnitude better performance than MiniONN on CIFAR-10. This motivated a shift back to HE for linear layers moving forward.

CrypTFlow2 [35] introduced a new protocol for the millionaires' problem (millionaires') leveraging the state-of-the-art optimizations and techniques for IKNP-style OT extension [36], [37]. It features highly efficient OT-based implementations of all nonlinear operations over $\mathbb{Z}_P$ as well as $\mathbb{Z}_{2^b}$ using this millionaires' protocol. The authors observe that protocols for $\mathbb{Z}_P$ are more expensive than the corresponding protocols for $\mathbb{Z}_{2^b}$, but in order to take advantage of the gains offered by HE they implement and integrate protocols for nonlinear operations over $\mathbb{Z}_P$ with Gazelle's HE protocols. CrypTFlow2 outperforms GC protocols for nonlinear operations in $\mathbb{Z}_P$ and $\mathbb{Z}_{2^b}$ by an order of magnitude and it is the first to show inference on ImageNet [20]

Cheetah [38] implements brand new algorithms for linear algebra operations over $\mathbb{Z}_{2^b}$ using HE. It builds on the observation that HE multiplications, which are basically polynomial multiplications, implicitly compute vector dot-products across the coefficients of both operands. To capitalize on this, Cheetah implements an encoding scheme that bypasses the

[3]Does not require any model customization or finetuning.

traditional encoding space of schemes like BFV/BGV and instead encodes messages directly into the coefficients of the polynomials. The messages are placed strategically within the polynomials such that one vector dot product is computed with a single polynomial multiplication. This strategy results in a sparse output with very few coefficients containing the actual result. Cheetah addresses this by extracting relevant coefficients from the output. Cheetah's protocols for linear algebra do not involve any HE rotations (slowest operation in HE) and outperform Gazelle's protocols by $5\times$ in computation and communication.

CrypTFlow2's algorithms with silent OT primitives derived from Ferret Silent OT [18] lay the foundation for all of Cheetah's protocols for nonlinear operations over $\mathbb{Z}_{2^b}$. To alleviate the computation overhead of silent OT, Cheetah implements a 1 bit approximate truncation particularly optimized for scenarios where the secret value is known to be positive. Truncation is delayed until after ReLU instead of performing it right after convolution or matrix multiplication to take advantage of the new optimized protocol. Cheetah achieves $3\times$ faster E2E runtimes with less than $10\times$ communication over CrypTFlow2.

HELiKs [15] is the latest in the line of works targetting secure linear algebra operations on high dimensional data levering HE. It presents new protocols for secure linear algebra operations HE over $\mathbb{Z}_P$ that outperform corresponding protocols in Cheetah in terms of both computation and communication for the same precision, while strictly adhering to the definitions of the HE schemes used. Cheetah's HE protocols for linear operations generate very sparse HE results, the coefficient extraction process deviates from HE scheme definitions and the final outputs generated by the protocols bear no similarity in structure compared to their corresponding inputs. Cheetah's HE results cannot be readily used by the server for any subsequent HE operations (if need be).

HELiKs works with modular kernels that maintain the same encoding format of the input in the output. The performance of these kernels is significantly improved by taking advantage of many mathematical optimizations in HE computation, such as noise growth reduction, 1-step rotations, NTT preprocessing, tiling for large inputs, and symmetric key encryption. HELiKs uses the secure computation protocols of CrypTFlow2 for nonlinear operations to perform secure inference. In the E2E evaluation, while it shows better runtimes than Cheetah (particularly for HE protocols), their net communication volume is high due to its reliance on CrypTFlow2 protocols with communication heavy IKNP-style OT extension.

Recently, there has also been a huge push in research for secure inference on transformer models [39], [40], [41], [42]. All of these transformer works make use of the aforementioned frameworks to serve as foundation protocols and build higher-level operations, e.g. softmax, GeLU, attention etc., using these foundation protocols.

III. Threat Model

The threat model for SecONNds assumes strictly two participating parties, the client and the service provider, exhibiting semi-honest (honest-but-curious) behavior. Both parties are expected to adhere to the protocol's steps while being potentially interested in deriving additional information from the exchange beyond their legitimate outputs. Under this model, the privacy of the client's input and the confidentiality of the service provider's neural network model, including its parameters and intermediate computations, are paramount. The model specifically aims to prevent the leakage of sensitive information to either party beyond what is minimally required for computation. This threat model excludes considerations of active adversaries who may deviate from protocol specifications or attempt side-channel attacks, focusing solely on data privacy against passive observation and analysis.

IV. Millionaires' Protocol

The Millionaires' problem was conceptualized by Yao [1] as two millionaires who want to learn who is richer without disclosing their wealth to each other. The solution for the millionaires' problem corresponds to securely evaluating a comparison operation involving two parties with one private input each. The current state-of-the-art algorithm for millionaires' with OT based secure evaluation was presented by Rathee et al. in CryptFlow2 [35]. The core part of their algorithm comes from the work of Garay et al. [43], who proposed a novel approach of decomposing large b bit inputs, x of one party ($\mathcal{P}_0$) and y of the other party ($\mathcal{P}_1$), into little-endian-ordered lists of $q = b/m$ m bit numbers each, $\{x_0, ..., x_{q-1}\}, \{y_0, ..., y_{q-1}\}$, and computing the result with an arithmetic circuit over m bit numbers. This computation involves first evaluating the inequality, $lt_{0,i} \in \{0, 1\}$, ($<$ or $>$ for evaluating $x < y$ or $x > y$ respectively) and the equality, $eq_{0,i} \in \{0, 1\}$, for the pair of m bit numbers $(x_i, y_i) \forall\, i \in [0, q)$, and then evaluating another arithmetic circuit that combines these results in a binary tree fashion with depth $\lceil \log q \rceil - 1$. The results of the q inequality and equality comparisons are laid out at the root level ($j = 0$), and for every level, $j \geq 1$, two bits, $lt_{j,i}$ and $eq_{j,i}$, are computed for all nodes i, following:

$$lt_{j,i} = lt_{j-1,2i} + eq_{j-1,2i} \times lt_{j-1,2i+1} \tag{1}$$

$$eq_{j,i} = eq_{j-1,2i} \times eq_{j-1,2i+1} \tag{2}$$

The equality comparison is skipped for the first node in every level, and the inequality result of the highest node is returned as the final output of the protocol. Garay et al. [43] used arithmetic circuits due to their choice of encryption primitives, such as the use of the Paillier cryptosystem [7] for secure integer multiplications. Earlier work by Blake et al. [44] showed how integer comparisons could be evaluated using OT. CryptFlow2's algorithm marries the OT based integer comparisons of Blake et al. [44] with the log-depth arithmetic circuit of Garay et al. [43] for combining the results of the integer comparisons. They improve the performance of the

integer comparisons by folding both the comparisons for one pair, inequality and equality, into one call to $\binom{2^m}{1}$–OT_2 with a total communication cost of $q(\lambda + 2^{m+1})$ bits for the q pairs. Since the results of the integer comparisons are secret shared bits, they implement a boolean version of the log-depth circuit of Garay et al. [43], by replacing $+$ with bit-XOR ($\oplus$) and $\times$ with bit-AND ($\wedge$). They evaluated $\wedge$–gates in this boolean circuit with secure-AND functionality $\mathcal{F}_{\mathsf{AND}}$ realized through bit-triples [28]. They optimize bit-triple generation with correlated bit-triples, observing that for the nodes that output 2 bits ($j \geq 1$, $i \geq 1$), the 2 calls to $\mathcal{F}_{\mathsf{AND}}$ share one operand.

The total communication cost to generate the triples is $(\lceil \log q \rceil - 1)(\lambda + 16) + (q - \lceil \log q \rceil)(\lambda + 8)$, and the total cost to evaluate the boolean circuit is the sum of the triple generation cost and $4(q-1) + 4(q - \lceil \log q \rceil)$ to share the correction bits (4 per $\mathcal{F}_{\mathsf{AND}}$). Huang et al. [38] presented an optimized version by porting the OT primitives to silent OT extension [18]. This reduced the communication cost to $q \times (2^{m+1} + m)$ for the q calls to $\binom{2^m}{1}$–OT_2 and only to $4(q-1) + 4(q - \lceil \log q \rceil)$ for the boolean circuit. They generate a bit-triple from $\binom{2}{1}$–ROT_1 following the strategy presented by Asharov et al. [37], and since their OTs are based on silent OT extension, $\binom{2}{1}$–ROT_1 is almost free in terms of communication. Both Cheetah and CryptFlow2 agree on setting $m = 4$ for the best performance. Consider the case of $m = 4$ vs. $m = 1$ with silent OT extension, the communication cost of m bit integer comparisons is: $9b$ for $m = 4$ and $5b$ for $m = 1$; while the cost of evaluating the boolean circuit is: $2b + 4 - 4\lceil \log b \rceil$ for $m = 4$ and $8b - 4 - 4\lceil \log b \rceil$ for $m = 1$. The total communication cost is slightly less than $11b$ for $m = 4$ and less than $13b$ for $m = 1$, and therefore setting $m = 1$ incurs approximately 18% overhead in terms of communication compared to setting $m = 4$.

When it comes to the computation cost, we observe that virtually all of it comes from performing $\binom{2^m}{1}$–OT_2's for integer comparisons and $\binom{2}{1}$–ROT_1's for triple generation. For 2^{13} comparisons of 32 bit numbers, the time taken to perform $\binom{2^m}{1}$–OT_2's is around 90ms for $m = 1$ and $m = 4$, the time taken for $\binom{2}{1}$–ROT_1's is around 110ms for $m = 1$ and 15ms for $m = 4$. Straightaway, one can speed up the online time (query response) of the protocol by generating the triples offline (before a query is requested). However, for the optimal setting of $m = 4$, this would only result in a mere acceleration of 16%. In the following subsection, we show how to completely replace the calls to $\binom{2^m}{1}$–OT_2 with $\binom{2}{1}$–ROT_1 and later show how to efficiently move the triple generation to the offline phase.

A. Fully-Boolean Algorithm

For the case of $m = 1$, observe that $\mathbb{1}\{x_i = y_i\} = (1 \oplus x_i) \oplus y_i$ and $\mathbb{1}\{x_i < y_i\} = (1 \oplus x_i) \wedge y_i$. Evaluating the equality comparisons for the bit pairs is free, $\mathcal{P}_0$ sets its share of the equals result $\langle eq_{0,i} \rangle_2^0 = 1 \oplus x_i$ and $\mathcal{P}_1$ sets its share of the equals result $\langle eq_{0,i} \rangle_2^1 = y_i$, and no communication is required. For inequalities, both parties make one call to $\mathcal{F}_{\mathsf{AND}}$ for each bit where $\mathcal{P}_0$ inputs $1 \oplus x_i$, $\mathcal{P}_1$ inputs y_i and set their shares of $lt_{0,i}$ to the output of $\mathcal{F}_{\mathsf{AND}}$. This approach only uses $\binom{2}{1}$–ROT_1 for the triple generation in $\mathcal{F}_{\mathsf{AND}}$ and takes less than 40 ms for 2^{13} comparisons with 32 bit numbers. It requires a communication of 4 bits for each $\mathcal{F}_{\mathsf{AND}}$ and a total communication of $4b$ bits.

Algorithm 1: $\mathcal{F}_{\mathsf{MILL}}$ Millionaires' Protocol

Input: Data bitwidth b ; Inequality $g : \{0,1\} \rightarrow \{<, >\}$

Input i_p

Output: Output secret share $\langle o \rangle_{2^b}^p$

for $i = 0$ **to** $b - 1$ **do**

/* Bit Extraction & Share Generation */

$\langle b_0 \rangle_2^p = \Big(\big(i_p / 2^i \bmod 2 \big) \oplus g' \Big) \wedge p'$

$\langle b_1 \rangle_2^p = \Big(\big(i_p / 2^i \bmod 2 \big) \oplus g \Big) \wedge p$

/* Bit Equality & Inequality Comparisons */

$\langle \mathbf{b_{eql}} \rangle_2^p[i] = \langle b_0 \rangle_2^p \oplus \langle b_1 \rangle_2^p$

$\langle \mathbf{b_{l/g}} \rangle_2^p[i] = \mathcal{F}_{\mathsf{AND}} \Big(\langle b_0 \rangle_2^p, \langle b_1 \rangle_2^p \Big)$

/* Combining Bit Results */

for $i = 0$ **to** $b - 2$ **do**

$\langle b_{and} \rangle_2^p = \mathcal{F}_{\mathsf{AND}} \Big(\langle \mathbf{b_{eql}} \rangle_2^p[i+1], \langle \mathbf{b_{l/g}} \rangle_2^p[i] \Big)$

$\langle \mathbf{b_{l/g}} \rangle_2^p[i+1] = \langle \mathbf{b_{l/g}} \rangle_2^p[i+1] \oplus \langle b_{and} \rangle_2^p$

$\langle o \rangle_2^p = \langle \mathbf{b_{l/g}} \rangle_2^p[b-1]$

TABLE I
COMMUNICATION COSTS AND RUNTIME OF MILLIONAIRES' PROTOCOL

Protocol	Communication	Runtime $\big(2^{13}$ calls$\big)$
Cheetah (m = 1)	$13b - 4 - 4\lceil \log b \rceil$	≈ 200 ms
Cheetah (m = 4)	$11b + 4 - 4\lceil \log b \rceil$	≈ 105 ms
SecONNds (ours)	$< 8b$	$< \big(70 + \mathbf{5}\big)$ ms Offline + Online

Following the log-depth approach of Garay et al. [43] for the later phase of the computation, we get a total communication cost of less than $12b$ bits for the millionaires' protocol. Although this is better than $16b$ in the case of using $\binom{2^m}{1}$–OT_2 for $m = 1$, it is still higher than $11b$ in the optimal setting of $m = 4$. Observe that, alternately to the log-depth strategy, we can also combine the integer comparison results serially in the following manner:

$$lt_0 = 0 \tag{3}$$

$$lt_i = lt_{i-1} \oplus lt_{0,i} \oplus (eq_{0,i} \wedge lt_{0,i-1}) \text{ for } i \in [1, b) \tag{4}$$

Following this linear strategy, the final result is produced

in the value lt_{q-1}, and the overall computation requires only $b-1$ calls to $\mathcal{F}_{\mathsf{AND}}$ which is roughly half of $2b-1-\lceil \log b \rceil$ in the case of the log-depth strategy with $m=1$. Thus, this linear strategy incurs a communication cost of only $4(b-1)$, bringing the total communication footprint of the millionaires' protocol to under $8b$ bits, which is 27% lower than the $11b$ bits cost of the $\binom{2^m}{1}$–OT_2 version with the optimal setting of $m=4$. This approach also requires half as many calls to $\binom{2}{1}$–ROT_1 and takes less than 35 ms for 2^{13} comparisons with 32 bit numbers.

In Table I, we show the total cost of the millionaires' protocol implemented in SecONNds, the runtimes are reported for 2^{13} runs with $b=32$. The total computation time for our new fully Boolean algorithm for the millionaires' protocol is under 75 ms with online triple generation, which is 28% less than the $\binom{2^m}{1}$–OT_2 version with the optimal setting of $m=4$. Note that while the linear strategy halves the communication footprint as well as the computation cost, it incurs an exponential increase in the number of rounds. Application developers can implement a simple toggle to switch between both strategies depending on available network resources to ensure the best quality of service.

B. Chuncked Triple Generation

To efficiently generate bit triples with the silent OT extension and to safely shift the triple generation to the offline phase, we implement a Chuncked triple generator with an internal buffer, inspired by the PRNG implementation in the cryptoTools library [45]. The triple generator automatically generates enough triples in chunks of fixed size to fill its buffer, as soon as a network connection with a user is established. When a query is requested, all underlying protocols make use of the `get` functionality of the triple generator to access the preprocessed triples. The triple generator automatically generates new triples and refills the buffer when it is exhausted during the online computation. In case a protocol requests for a volume of triples larger than the buffer size, the buffer size is incremented, and the generator generates new triples in chunks of fixed size to fill the buffer.

Chuncking increases the complexity of the communication for n calls to $\binom{2}{1}$–ROT_1 from $O(\log n)$ to a sublinear $O\left(\frac{n}{m}\log m\right)$ where m is the size of one chunk. For any large m, the communication footprint of our chuncking strategy is fairly comparable to the naive approach of generating n $\binom{2}{1}$–ROT_1's in one-shot. The real advantage of the chuncking strategy comes in terms of computation time, which is actually the main concern with silent OT. The computational complexity involved in naively generating n $\binom{2}{1}$–ROT_1's is $O(n^2)$, arising from the matrix multiplication involved in the LPN encoding phase of silent OT. With the chunking strategy, the computation complexity is significantly reduced to $O(nm)$ which is now linear in n.

V. Nonlinear Operations

In this section, we review the protocols within SecONNds for nonlinear operations involved in quantized Convolutional Neural Networks (CNNs): Truncation, ReLU, and Max Pooling. Although Average Pooling and ArgMax are also integral to completing the architecture of neural networks, their impact on the overall performance of the network is relatively negligible because they are called very few times throughout the network's operation. Therefore, we will briefly review these functions in the Appendix, focusing the main discussion on operations that significantly affect system performance.

Algorithm 2: $\mathcal{F}_{\mathsf{ReLU}}$ ReLU

Input: Input secret share $\langle i \rangle^p_{2^b}$

Output: Output secret share $\langle o \rangle^p_{2^b}$

$\mathsf{MSB}\left(\langle i \rangle^p_{2^b}\right) = \langle i \rangle^p_{2^b} / 2^{b-1}$

$\left|\langle i \rangle^p_{2^b}\right| = \langle i \rangle^p_{2^b} - \mathsf{MSB}\left(\langle i \rangle^p_{2^b}\right) \cdot 2^{b-1}$

$i_{mill} = (-1)^p \left|\langle i \rangle^p_{2^b}\right| + p \cdot \left(2^{b-1} - 1\right)$

$\langle w \rangle^p_2 = \mathcal{F}_{\mathsf{MILL}}\left(b-1, 1, i_{mill}\right)$

$\langle i_{drelu} \rangle^p_2 = \mathsf{MSB}\left(\langle i \rangle^p_{2^b}\right) \oplus \langle w \rangle^p_2 \oplus p'$ // dReLU result

$\delta = (1 - 2 \cdot \langle i_{drelu} \rangle^p_2) \cdot \langle i \rangle^p_{2^b}$ // Delta for COT

$c = \langle i_{drelu} \rangle^p_2$ // Choice for COT

$m_s = \binom{2}{1}$–COT_b–$\mathsf{send}(\delta)$

$m_r = \binom{2}{1}$–COT_b–$\mathsf{receive}(c)$

$\langle o \rangle^p_{2^b} = \left(\langle i \rangle^p_{2^b} \cdot \langle i_{drelu} \rangle^p_2 + m_r - m_s\right) \bmod 2^b$

A. ReLU

The function ReLU (Rectified Linear Unit) is a widely used activation function in neural networks, defined as $\mathsf{ReLU}(i) = \max(0, i)$. In SecONNds, we employ Cheetah's Silent OT-based implementation of the CrypTFlow2 protocol with the millionaires' protocol described in Section IV. This protocol, denoted as $\mathcal{F}_{\mathsf{ReLU}}$, is shown in Algorithm 2.

The protocol takes as input the secret shares of the activations entering the ReLU layer and returns the secret shares of the ReLU result. The protocol first evaluates $d\mathsf{ReLU} = \mathbb{1}\{i > 0\}$ and returns fresh secret shares of i if $d\mathsf{ReLU}$ is 1 and secret shares of 0 if $d\mathsf{ReLU}$ is zero. Observe that $i > 0$ in $\mathbb{Z}_{2^b}$ corresponds to $i < 2^{b-1}$, which is equivalent to:

$$\left(\mathsf{MSB}\left(\langle i \rangle^0_{2^b}\right) + \mathsf{MSB}\left(\langle i \rangle^1_{2^b}\right)\right) \cdot 2^{b-1} + \left|\langle i \rangle^0_{2^b}\right| + \left|\langle i \rangle^1_{2^b}\right| < 2^{b-1}$$

This inequality depends only on the sum of the absolute values of the shares wrapping around the maximum absolute value in the ring, $2^{b-1}-1$, denoted by the bit w in Algorithm 2, and the equality of the most significant bits (MSB) of the shares. Particulary, it holds only if w is 0 and both MSBs are equal, or if w is 1 and both MSBs are not equal:

$$d\mathsf{ReLU} = \mathsf{MSB}\left(\langle i \rangle^0_{2^b}\right) \oplus \mathsf{MSB}\left(\langle i \rangle^1_{2^b}\right) \oplus \langle w \rangle^0_2 \oplus \langle w \rangle^1_2 \oplus 1$$

Algorithm 3: $\mathcal{F}_{\mathsf{MaxPool}}$ Max Pooling

Input: Flattened window size w

Input secret share array $\langle \mathbf{i} \rangle_{2^b}^p$ of size w

Output: output secret share $\langle o \rangle_{2^b}^p$

$\langle o \rangle_{2^b}^p = \langle \mathbf{i} \rangle_{2^b}^p[0]$

for $k = 1$ **to** $w - 1$ **do**

  $\langle o \rangle_{2^b}^p = \mathcal{F}_{\mathsf{ReLU}}\Big(\langle o \rangle_{2^b}^p - \langle \mathbf{i} \rangle_{2^b}^p[k]\Big) + \langle \mathbf{i} \rangle_{2^b}^p[k]$

Here, the wrap bit w is securely computed using the millionaires' protocol described in Section IV. After computing $d\mathsf{ReLU}$, the protocol uses a secure multiplexer functionality, MUX, realized with two calls to $\binom{2}{1}$–COT_b. The $d\mathsf{ReLU}$ result serves as the selection bit. If the $d\mathsf{ReLU}$ result is 1, the MUX outputs new shares of the input; if the $d\mathsf{ReLU}$ result is 0, the MUX outputs shares of zero. This ensures that only the activated inputs are passed through while the others are set to zero.

The core secure computation operations involved in $\mathcal{F}_{\mathsf{ReLU}}$ are the $2b - 3$ calls to $\mathcal{F}_{\mathsf{AND}}$ in $\mathcal{F}_{\mathsf{MILL}}$ and the 2 calls to $\binom{2}{1}$–COT_b for MUX.

B. Max Pooling

Pooling is a fundamental operation in convolutional neural networks aimed at downsampling input feature maps to highlight dominant features. Max Pooling achieves this by extracting the maximum value from each specified segment of the input array. In SecONNds, Max Pooling is implemented securely through a protocol called $\mathcal{F}_{\mathsf{MaxPool}}$, which is outlined in Algorithm 3.

The core functionality of the $\mathcal{F}_{\mathsf{MaxPool}}$ protocol involves taking secret shares of an input layer and securely determining the maximum values for designated regions or windows. The protocol initializes the presumed maximum with the first element of each window and securely iterates through the remaining elements to find the actual maximum. Each comparison is performed with one call to $\mathcal{F}_{\mathsf{ReLU}}$, totaling w calls to compute one output element where w is the flattened window size for pooling.

C. Truncation

In the context of fixed-point computation, truncation is a crucial operation to prevent the data scale from escalating after multiplications. We present the protocol employed in SecONNds, denoted as $\mathcal{F}_{\mathsf{Trunc}}$, in Algorithm 4. The core concept of this approach is to represent the data in its unsigned form as follows:

$$w = \mathbb{1}\Big\{\langle i \rangle_{2^b}^0 + \langle i \rangle_{2^b}^1 > 2^b - 1\Big\}$$
$$i = \langle i \rangle_{2^b}^0 + \langle i \rangle_{2^b}^1 - w \cdot 2^b$$
$$i/2^s \approx \langle i \rangle_{2^b}^0/2^s + \langle i \rangle_{2^b}^1/2^s - w \cdot 2^{b-s}$$

Algorithm 4: $\mathcal{F}_{\mathsf{Trunc}}$ Truncation

Input: Right shift amount s ;

Bit i_{msb} indicating if $\mathsf{MSB}(i)$ is known;

Input secret share $\langle i \rangle_{2^b}^p$ where $i < 2^{b-1}$

Output: Output secret share $\langle o \rangle_{2^b}^p$

$\mathsf{MSB}\left(\langle i \rangle_{2^b}^p\right) = \langle i \rangle_{2^b}^p / 2^{b-1}$

/* Wrap Bit Computation */

if i_{msb} **then**

  $\langle w \rangle_2^p = \mathcal{F}_{\mathsf{AND}}\Big(p \cdot \mathsf{MSB}\Big(\langle i \rangle_{2^b}^p\Big),\, p' \cdot \mathsf{MSB}\Big(\langle i \rangle_{2^b}^p\Big)\Big)$

else

  $\langle w \rangle_2^p = \mathcal{F}_{\mathsf{MILL}}\Big(b, 1, \Big[(-1)^p \langle i \rangle_{2^b}^p + p \cdot \left(2^b - 1\right)\Big]\Big)$

/* Wrap B2A Share Conversion */

if $p = 0$ **then**

  $\delta = -2 \cdot \langle w \rangle_2^p$ // Delta for COT

  $m_s = \binom{2}{1}$–COT_b–$\mathsf{send}(\delta)$

  $\langle w \rangle_{2^b}^p = \langle w \rangle_2^p - m_s$

else

  $c = \langle w \rangle_2^p$ // Choice for COT

  $m_r = \binom{2}{1}$–COT_b–$\mathsf{receive}(c)$

  $\langle w \rangle_{2^b}^p = \langle w \rangle_2^p + m_r$

/* Final Truncation Result */

$\langle o \rangle_{2^b}^p = \langle i \rangle_{2^b}^p / 2^s - \langle w \rangle_{2^b}^p \cdot 2^{b-s}$

This computation is approximate, as it does not account for potential carries from the truncated bits in the secret shares. However, this introduces an error only in the least significant bit of the result, and previous work [46], [38] has shown that neural networks are highly tolerant to this particular error in truncation, with negligible impact on performance. The wrap bit w is computed using $\mathcal{F}_{\mathsf{MILL}}$, but when the sign of the secret value is known, such as post-ReLU when the values are positive, $w = \mathsf{MSB}\Big(\langle i \rangle_{2^b}^0\Big) \wedge \mathsf{MSB}\Big(\langle i \rangle_{2^b}^1\Big)$ and can be computed with a single call to $\mathcal{F}_{\mathsf{AND}}$ using offline bit triples. In total, the main secure computation operations in this protocol are $2b - 1$ calls to $\mathcal{F}_{\mathsf{AND}}$ for $\mathcal{F}_{\mathsf{MILL}}$ and one call to $\binom{2}{1}$–COT_b to convert the secret shares of w from binary to arithmetic, or just one $\mathcal{F}_{\mathsf{AND}}$ and one $\binom{2}{1}$–COT_b if the MSB of the secret share is known.

D. Secret-sharing in $\mathbb{Z}_{2^b}$ vs. $\mathbb{Z}_P$

The choice of the ring for secret sharing, $\mathbb{Z}_{2^b}$ (a power of two) or $\mathbb{Z}_P$ (where P is prime), significantly influences

the implementation and performance of nonlinear operations. In $\mathbb{Z}_P$, any protocol requiring a comparison (like wrap bit computation) must also check if the secret overflows P and must make provision for handling it. On the other hand, operations in $\mathbb{Z}_{2^b}$ benefit from natural alignment with binary systems, the boolean circuit in $\mathcal{F}_{\mathsf{MILL}}$ also handles only an explicitly specified bitwidth, which doesn't require any overflow management.

VI. Linear Operations

The typical linear layers in CNNs consist of the convolution layers, fully-connected/matrix multiplication layers, and batch normalization. The bulk of linear operations come from convolutions with matrix multiplications that appear only at the very end of the CNN. Batch normalization typically appears after convolutions, and it is common practice to fuse batch normalization with convolution [47] during inference.

A. HE Kernels

Linear algebra operations can be composed from vector multiplications, rotations, and additions, all of which are supported by modern HE schemes for the plaintext space Z_P^N where P is the HE plaintext modulus prime. HELiKs [15] offers state-of-the-art kernels that compose these operations in the most efficient manner. At the core of these protocols is an iterative algorithm where each iteration involves a HE multiplication of the input vector with the weights, accumulating the product into the previous iteration's result, and finally a HE rotation to shift the accumulated value to adjust for the next iteration. This strategy significantly reduces the number of HE rotations required. HELiKs further boosts the performance by preprocessing the weights with NTT which leads to a very fast online runtime. Although HELiKs offer cutting-edge performance for linear algebra operations, for secure inference, it necessitates the use of Z_P for nonlinear operations or the use of a share conversion protocol to convert secret shares from Z_P to secret shares in Z_{2^b}.

Cheetah's HE kernels operate in the plaintext space $Z_{2^b}^N$. It encodes secret data into the polynomial coefficients, enabling polynomial multiplications and additions to secret data through HE multiplications and additions. Cheetah's HE kernels compute the linear algebra operations purely through iterative HE multiply-accumulate (MAC) operations without any HE rotations. They produce sparse results in HE, and then extract just the relevant coefficients from these results to produce the final results for the linear algebra operation.

B. NTT preprocessing

We optimize Cheetah's HE kernels for $Z_{2^b}^N$ with NTT-preprocessing. HE multiplications involve polynomial multiplication, which is $O(N^2)$ in the computation complexity. The polynomial degree modulus N is typically of the order of thousands or higher and induces a very long runtime for multiplication. HE libraries typically optimize this operation by transforming both operands with NTT, performing a Hadamard product on the transformed operands, and transforming the product back to its natural representation with iNTT (inverse NTT). Every HE multiplication involves two NTT operations and one iNTT operation, both involving a complexity of $O(N \log N)$.

Balla and Koushanfar [15] observed that, for linear algebra operations, most of the calls to HE multiplication share one of the operands (input vector) and the other operand (weights) are static (reused over multiple queries) and known to the server (computing party). In SecONNds, the weights are automatically preprocessed with NTT during encoding and are always maintained in the NTT representations. During the online query, on receipt of the input ciphertexts, the server first transforms each ciphertext with NTT, performs all HE MAC operations in NTT, and only transforms the final HE results back from NTT. Since these HE results are sparse, only coefficients that contain the elements of the output vector are extracted and sent back to the client for decryption.

C. GPU Acceleration

HE is a prime candidate for hardware acceleration because it does not require any communication to perform secure computations. The one-ended computation of HE protocols requires just the computing party to have access to the hardware accelerator. Today, in the context of remote cloud computation, it is very common for servers to possess GPUs. Moreover, polynomial data types are usually represented with list data structures, which are perfect for GPU-based SIMD computations.

Troy [19] is a new software library that implements the SEAL HE Library [48] in CUDA [49]. SecONNds employs the HE evaluator from Troy to implement the server's HE computations on GPU. All of client's computation is performed with the standard SEAL Library, the new GPU implementations do not handle any secret key related operations and purely compute only on already encrypted data.

VII. Evaluation

SecONNds is implemented in the OpenCheetah [50] variant of the Secure and Correct Inference (SCI) Library [51]. We present two versions, SecONNds for $\mathbb{Z}_2^b$ and SecONNds–P for $\mathbb{Z}_P$, where P is a BFV-SIMD plaintext prime modulus. SecONNds offers the best performance and uses 1 bit-approximate truncation. SecONNds–P employs faithful truncation and returns bit-exact results compared to the plaintext model.

For all our evaluations, we outfit all frameworks being compared with: Ferret Silent OT Extension [18] from the EMP-OT Library [52]; BFV HE scheme [11], [12] from the SEAL Library [48]; and server-side GPU acceleration with the HE Evaluator from the Troy Library [19]. All CPU operations of both parties are performed on 16 threads of an *Intel Xeon Gold 6338* processor, supplemented by 1 TB of RAM and utilizing both the *AES-NI* and *AVX-512* instruction set extensions. The server-side GPU evaluations are performed on a *NVIDIA RTX A6000* system.

TABLE II
OPERATION RUNTIMES (IN SECONDS) AND COMMUNICATION (IN MIB) FOR NONLINEAR OPERATIONS

Operation	CrypTFlow2		HELiKs		Cheetah		SecONNds–P		**SecONNds**	
	Time	**Comm.**	**Time**	**Comm.**	**Time**	**Comm.**	**Time**	**Comm.**	**Time**	**Comm.**
ReLU	8.77	302.39	9.03	311.46	3.84	49.97	3.17	186.19	**0.24**	**47.45**
Max Pooling	8.03	398.43	8.27	410.38	4.46	134.83	1.94	247.05	**0.20**	**121.27**
Truncation	4.75	307.57	4.89	316.80	0.14	5.00	0.61	267.48	**0.06**	**4.86**
Avg Pooling	0.05	4.19	0.05	4.32	0.06	4.42	0.04	4.15	0.05	4.55
Arg Max	0.09	0.19	0.09	0.20	0.04	0.10	0.09	0.19	0.04	0.10

TABLE III
OPERATION RUNTIMES (IN SECONDS) AND COMMUNICATION (IN MIB) FOR CONVOLUTION

Metric	CrypTFlow2		HELiKs		Cheetah		SecONNds–P		**SecONNds**	
	CPU	**GPU**	**CPU**	**GPU**	**CPU**	**GPU**	**CPU**	**GPU**	**CPU**	**GPU**
Communication	551.35	-	130.14	-	204.84	-	**130.14**	-	204.84	-
Runtime	25.20	4.13	2.81	0.54	4.69	0.58	**2.80**	0.55	3.47	**0.45**

TABLE IV
OFFLINE TRIPLE GENERATION COSTS

Framework	# Triples	Runtime	Communication
SecONNds–P	51380528	17.62 s	43.08 MiB
SecONNds	21200624	5.59 s	17.24 MiB

TABLE V
OFFLINE NTT-PREPROCESSING ON CPU AND GPU

Framework	CPU	GPU
SecONNds–P (HELiKs)	13195 s	1.517 s
SecONNds	0.728 s	0.089 s

We use a SqueezeNet [21] CNN model in our evaluations. The SqueezeNet model is uniformly quantized to fixed-point with a bitwidth of 32 and scale of 12.

A. Offline Preprocessing

SecONNds uses offline triples for for nonlinear operations. In Table IV, we summarize the costs of generating the offline triples for one secure inference using SqueezeNet. As mentioned in Section V-D, nonlinear operations in $\mathbb{Z}_P$ are more expensive than the same in $\mathbb{Z}_{2^b}$, SecONNds–P requires $2.4\times$ more triples than SecONNds.

SecONNds performs NTT preprocessing on the model weights for fast online HE computation. This process does not require any secret key information and moreover it is completely query independent. The server can reuse the NTT preprocessed weights unless the model parameters are updated or modified.

B. Nonlinear layers

In Table II, we show the cumulative performance of the nonlinear protocols for all calls in a single secure inference on SqueezeNet. The bulk of the runtime comes from the calls to the millionaires' protocol as seen by the drop in the runtimes for SecONNds vs. Cheetah and SecONNds–P vs. HELiKs. Compared to Cheetah, SecONNds achieves a speedup of $17\times$ for ReLU, $24\times$ for Max pooling and $2.3\times$ for truncation. The communication performance of SecONNds is similar to that of Cheetah. SecONNds–P acheives speedups of $2.8\times$ for ReLU, $4.3\times$ for Max Pooling and $8\times$ for truncation over HELiKs/CrypTFlow2, while requiring $1.6\times$, $1.6\times$ and $1.1\times$ lower communication, respectively.

C. Linear Layers

In Table III, we show the performance of various schemes for convolution operations. SecONNds achieves a speedup of $1.4\times$ over Cheetah on CPU runtime and a speedup of $1.3\times$ on GPU due to NTT preprocessing. SecONNds–P achieves speedups of $1.7\times$ on CPU runtime and $1.1\times$ on GPU runtime over Cheetah. Overall, SecONNds performs the best in terms of GPU runtimes, while SecONNds–P (HELiKs) performs the best on a CPU back-end. Since SecONNds does not involve any HE rotations, it enjoys a higher acceleration on GPU[4]

D. E2E Evaluation

This section evaluates the end-to-end (E2E) performance of the SecONNds framework, focusing on CPU and GPU runtimes, along with communication overhead for several operations including Convolution, ReLU, Max Pooling, Average Pooling, Truncation, and Argmax. These metrics are depicted in Figures 2, 3, and 4.

CPU Runtime: Illustrated in Figure 2, SecONNds leverages the new Millionaires' protocol with offline triples to enhance

[4]HE Rotation are only $3.2\times$ faster on GPU vs. CPU in Troy [19].

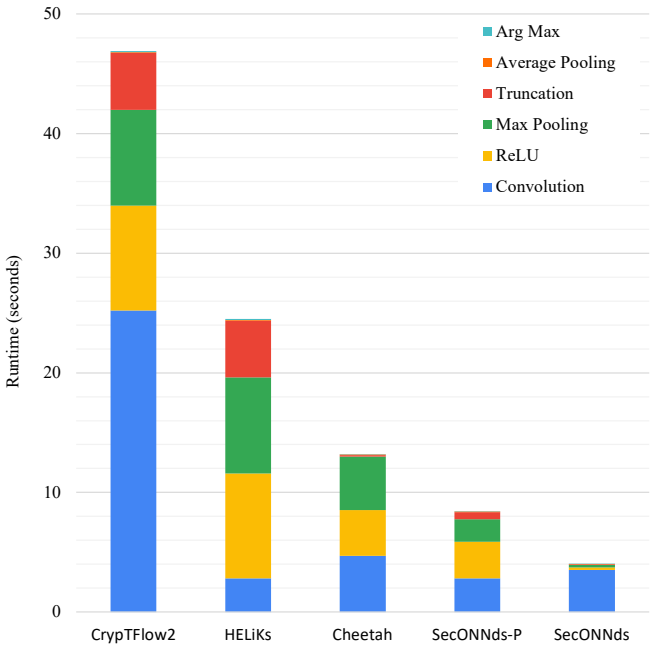

Fig. 2. CPU Runtime Comparison for Online Operations

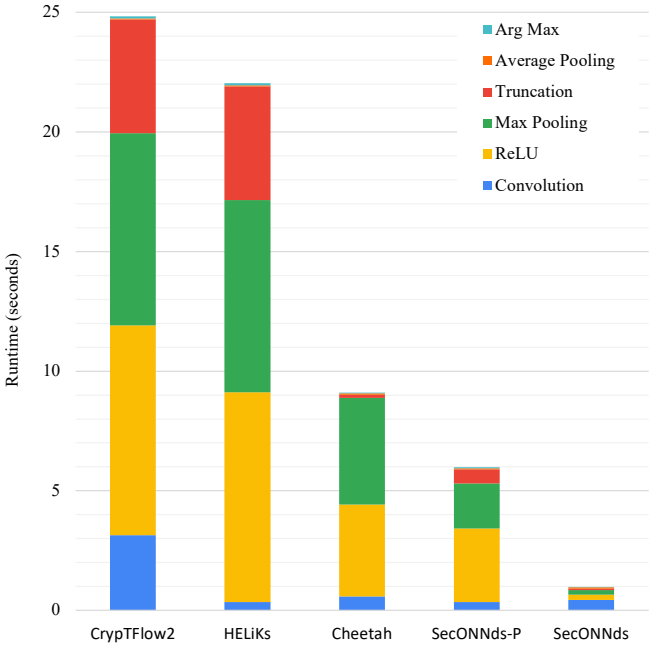

Fig. 3. GPU Runtime Comparison for Online Operations

CPU efficiency, offloading over 86% of computation to the server and involving the client predominantly in nonlinear operations. Consequently, SecONNds outperforms Cheetah by a factor of 3.3 and SecONNds–P surpasses HELiKs by 2.9×.

GPU Runtime: As shown in Figure 3, HE operations benefit significantly from GPU acceleration. The absence of

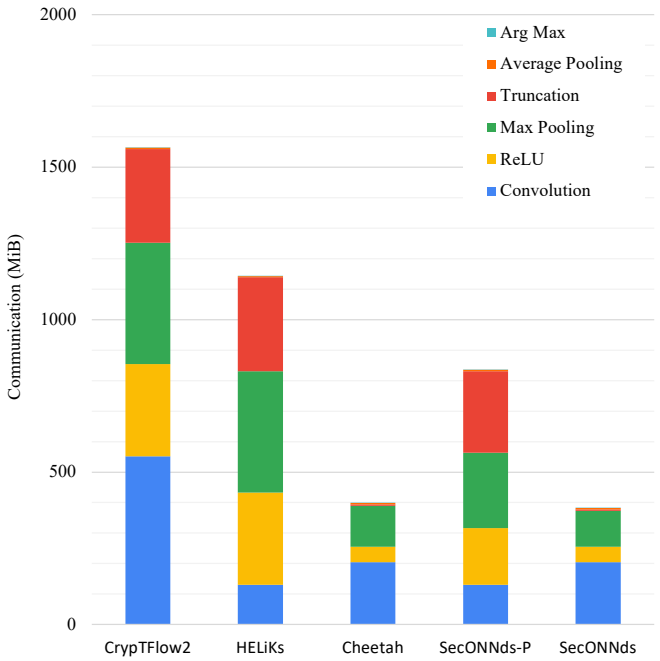

Fig. 4. Communication Overhead for Online Operations

HE rotations in SecONNds and Cheetah allows for higher performance gains. SecONNds achieves a ninefold increase over Cheetah, while SecONNds–P realizes a 3.7-fold improvement over HELiKs.

Communication Overhead: Figure 4 outlines the communication costs. SecONNds reduces communication overhead by 5% compared to Cheetah, and SecONNds–P reduces it by 27% relative to HELiKs.

Overall, SecONNds achieves E2E times of 4.05 seconds on CPU and 0.98 seconds on GPU, with 382 MiB of communication. Conversely, SecONNds–P posts times of 8.44 seconds on CPU and 5.97 seconds on GPU, requiring 835 MiB of communication.

VIII. Conclusion

In this paper, we introduced SecONNds, a novel framework for secure inference that leverages advanced cryptographic techniques and hardware acceleration to achieve efficient and secure neural network inference. By introducing a fully-Boolean Millionaires' protocol and leveraging NTT preprocessing with GPU acceleration, SecONNds achieves significant improvements in both computation and communication costs. The end-to-end evaluation demonstrates substantial performance gains in runtime over existing frameworks, achieving secure inference on ImageNet in under a second. Thus SecONNds provides an efficient security framework for outsourced neural network inference, offering a scalable solution for privacy-sensitive applications.

REFERENCES

[1] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164.

[2] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game, or a completeness theorem for protocols with honest majority," in *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, 2019, pp. 307–328.

[3] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2009, aAI3382729.

[4] C. Crépeau, J. van de Graaf, and A. Tapp, "Committed Oblivious Transfer and Private Multi-Party Computation," in *Advances in Cryptology — CRYPT0' 95*, D. Coppersmith, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 110–123.

[5] R. Cramer, I. Damgård, and U. Maurer, "General Secure Multi-Party Computation from any Linear Secret Sharing Scheme," Cryptology ePrint Archive, Paper 2000/037, 2000, https://eprint.iacr.org/2000/037. [Online]. Available: https://eprint.iacr.org/2000/037

[6] R. L. Rivest, L. Adleman, M. L. Dertouzos *et al.*, "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[7] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology — EUROCRYPT '99*, J. Stern, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 223–238.

[8] L. Ducas and D. Micciancio, "FHEW: Bootstrapping Homomorphic Encryption in less than a second," Cryptology ePrint Archive, Paper 2014/816, 2014, https://eprint.iacr.org/2014/816. [Online]. Available: https://eprint.iacr.org/2014/816

[9] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "TFHE: Fast Fully Homomorphic Encryption over the Torus," Cryptology ePrint Archive, Paper 2018/421, 2018, https://eprint.iacr.org/2018/421. [Online]. Available: https://eprint.iacr.org/2018/421

[10] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "Fully Homomorphic Encryption without Bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, 2011, p. 309–325. [Online]. Available: https://eprint.iacr.org/2011/277.pdf

[11] Z. Brakerski and V. Vaikuntanathan, "Efficient Fully Homomorphic Encryption from (Standard) LWE," in *Proceedings of the 52nd Annual Symposium on Foundations of Computer Science*, 2011, p. 97–106. [Online]. Available: https://eprint.iacr.org/2011/344.pdf

[12] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption," Cryptology ePrint Archive, Paper 2012/144, 2012, https://eprint.iacr.org/2012/144. [Online]. Available: https://eprint.iacr.org/2012/144

[13] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," in *Advances in Cryptology – ASIACRYPT 2017*, 2017, p. 409–437. [Online]. Available: https://eprint.iacr.org/2016/421.pdf

[14] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," 2016. [Online]. Available: https://arxiv.org/abs/1609.07061

[15] S. Balla and F. Koushanfar, "HELiKs: HE Linear Algebra Kernels for Secure Inference," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 2306–2320. [Online]. Available: https://doi.org/10.1145/3576915.3623136

[16] E. Boyle, G. Couteau, N. Gilboa, Y. Ishai, L. Kohl, and P. Scholl, "Efficient Pseudorandom Correlation Generators: Silent OT Extension and More," in *Advances in Cryptology – CRYPTO 2019 - 39th Annual International Cryptology Conference, Proceedings*, ser. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), D. Micciancio and A. Boldyreva, Eds. Germany: Springer Verlag, Jan. 2019, pp. 489–518.

[17] D. Beaver, "Correlated pseudorandomness and the complexity of private computations," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96'. New York, NY, USA: Association for Computing Machinery, 1996, p. 479–488. [Online]. Available: https://doi.org/10.1145/237814.237996

[18] K. Yang, C. Weng, X. Lan, J. Zhang, and X. Wang, "Ferret: Fast Extension for Correlated OT with Small Communication," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1607–1626. [Online]. Available: https://doi.org/10.1145/3372297.3417276

[19] Lightbulb, "Troy: Gpu implementation of bfv, ckks and bgv he schemes from seal," 2021. [Online]. Available: https://github.com/lightbulb128/troy

[20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[21] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size," 2016. [Online]. Available: https://arxiv.org/abs/1602.07360

[22] M. O. Rabin, "How To Exchange Secrets with Oblivious Transfer," Cryptology ePrint Archive, Paper 2005/187, 1981. [Online]. Available: https://eprint.iacr.org/2005/187

[23] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending Oblivious Transfers Efficiently," in *Advances in Cryptology - CRYPTO 2003*, D. Boneh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 145–161.

[24] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable pseudorandom functions and applications," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, 2013, pp. 669–684.

[25] D. Boneh and B. Waters, "Constrained pseudorandom functions and their applications," in *International conference on the theory and application of cryptology and information security*. Springer, 2013, pp. 280–300.

[26] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *Journal of the ACM (JACM)*, vol. 33, no. 4, pp. 792–807, 1986.

[27] A. Blum, M. Furst, M. Kearns, and R. J. Lipton, "Cryptographic primitives based on hard learning problems," in *Annual International Cryptology Conference*. Springer, 1993, pp. 278–291.

[28] D. Beaver, "Efficient Multiparty Protocols Using Circuit Randomization," in *Advances in Cryptology — CRYPTO '91*, J. Feigenbaum, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432.

[29] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," *J. ACM*, vol. 60, no. 6, nov 2013. [Online]. Available: https://doi.org/10.1145/2535925

[30] 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 *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger, Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 201–210. [Online]. Available: https://proceedings.mlr.press/v48/gilad-bachrach16.html

[31] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[32] 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*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 619–631. [Online]. Available: https://doi.org/10.1145/3133956.3134056

[33] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images," University of Toronto, Toronto, Ontario, Canada, Technical Report TR-2009, 2009. [Online]. Available: https://www.cs.toronto.edu/ kriz/learning-features-2009-TR.pdf

[34] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "GAZELLE: A Low Latency Framework for Secure Neural Network Inference," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, pp. 1651–1669. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[35] D. Rathee, M. Rathee, N. Kumar, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "CrypTFlow2: Practical 2-Party Secure Inference," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 325–342. [Online]. Available: https://doi.org/10.1145/3372297.3417274

[36] V. Kolesnikov and R. Kumaresan, "Improved ot extension for transferring short secrets," in *CRYPTO*.

Springer, 2013, pp. 54–70. [Online]. Available: https://www.iacr.org/archive/crypto2013/80420329/80420329.pdf
[37] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 535–548. [Online]. Available: https://doi.org/10.1145/2508859.2516738
[38] Z. Huang, W. jie Lu, C. Hong, and J. Ding, "Cheetah: Lean and Fast Secure Two-Party Deep Neural Network Inference," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 809–826. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/huang-zhicong
[39] D. Rathee, M. Rathee, R. K. K. Goli, D. Gupta, R. Sharma, N. Chandran, and A. Rastogi, "SIRNN: A math library for secure RNN inference," Cryptology ePrint Archive, Paper 2021/459, 2021, https://eprint.iacr.org/2021/459. [Online]. Available: https://eprint.iacr.org/2021/459
[40] M. Hao, H. Li, H. Chen, P. Xing, G. Xu, and T. Zhang, "Iron: Private Inference on Transformers," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 15 718–15 731.
[41] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "BOLT: Privacy-preserving, accurate and efficient inference for transformers," Cryptology ePrint Archive, Paper 2023/1893, 2023, https://eprint.iacr.org/2023/1893. [Online]. Available: https://eprint.iacr.org/2023/1893
[42] W. jie Lu, Z. Huang, Z. Gu, J. Li, J. Liu, C. Hong, K. Ren, T. Wei, and W. Chen, "BumbleBee: Secure two-party inference framework for large transformers," Cryptology ePrint Archive, Paper 2023/1678, 2023, https://eprint.iacr.org/2023/1678. [Online]. Available: https://eprint.iacr.org/2023/1678
[43] J. Garay, B. Schoenmakers, and J. Villegas, "Practical and Secure Solutions for Integer Comparison," in *Public Key Cryptography – PKC 2007*, T. Okamoto and X. Wang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 330–342.
[44] I. F. Blake and V. Kolesnikov, "Strong Conditional Oblivious Transfer and Computing on Intervals," in *Advances in Cryptology - ASIACRYPT 2004*, P. J. Lee, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 515–529.
[45] P. Rindal, "cryptoTools," https://github.com/ladnir/cryptoTools, 2024, accessed: 2024-06-04.
[46] A. Dalskov, D. Escudero, and M. Keller, "Secure Evaluation of Quantized Neural Networks," Cryptology ePrint Archive, Paper 2019/131, 2019, https://eprint.iacr.org/2019/131. [Online]. Available: https://eprint.iacr.org/2019/131
[47] N. Markus, "Fusing batch normalization and convolution in runtime," 2023, accessed: 2024-07-10. [Online]. Available: https://nenadmarkus.com/p/fusing-batchnorm-and-conv/
[48] Microsoft, "Microsoft SEAL (Version 4.0)," https://github.com/microsoft/SEAL/tree/4.0.0, 2020, accessed: 2024-07-07.
[49] NVIDIA Corporation, "CUDA Toolkit," 2021, version 11.0. [Online]. Available: https://developer.nvidia.com/cuda-toolkit
[50] Alibaba Gemini Lab, "OpenCheetah," https://github.com/Alibaba-Gemini-Lab/OpenCheetah, 2024, accessed: 2024-07-07.
[51] MPC-MSRI, "Secure and Correct Inference (SCI) Library," https://github.com/mpc-msri/EzPC/tree/master/SCI, 2024, accessed: 2024-07-07.
[52] X. Wang and Others, "EMP-OT Library," https://github.com/emp-toolkit/emp-ot, 2024, accessed: 2024-07-07.

APPENDIX

A. Secure AND

Algorithm 5: $\mathcal{F}_{\mathsf{AND}}$ And with Offline Bit Triples

Input: Input secret shares $\langle i_0 \rangle_2^p$ and $\langle i_1 \rangle_2^p$

Output: Output secret share $\langle o \rangle_2^p$

/* Triple Retrieval */

$\{\langle a \rangle_2^p, \langle b \rangle_2^p, \langle c \rangle_2^p\} = \mathsf{TripleGen.get}(1)$

/* Correction bit Computation */

$\langle e \rangle_2^p = \langle a \rangle_2^p \oplus \langle i_0 \rangle_2^p$; $\langle f \rangle_2^p = \langle b \rangle_2^p \oplus \langle i_1 \rangle_2^p$

/* Correction bit Reveal */

$\mathsf{Send}(\langle e \rangle_2^p)$; $\mathsf{Send}(\langle f \rangle_2^p)$

$\langle e \rangle_2^{p'} = \mathsf{Receive}()$; $\langle f \rangle_2^{p'} = \mathsf{Receive}()$

$e = \langle e \rangle_2^p \oplus \langle e \rangle_2^{p'}$; $f = \langle f \rangle_2^p \oplus \langle f \rangle_2^{p'}$

$\langle o \rangle_2^p = (p' \wedge e \wedge f) \oplus (e \wedge \langle b \rangle_2^p) \oplus (f \wedge \langle a \rangle_2^p) \oplus \langle c \rangle_2^p$

Algorithm 5 shows the protocol for the secure and functionality in SecONNds. This algorithm is a straightforward implementation of GMW [2] with preprocessed Beaver's Bit Triples.

B. Average Pooling

Algorithm 6: $\mathcal{F}_{\mathsf{AvgPool}}$ Average Pooling

Input: Output size n ; Flattened window size w

Input secret share array $\langle \mathbf{I} \rangle_{2^b}^p$ of size $n \times w$

Output: Flattened output secret share $\langle \mathbf{o} \rangle_{2^b}^p$

for $j = 0$ **to** $n - 1$ **do**
  $\langle \mathbf{o} \rangle_{2^b}^p[j] = \langle \mathbf{I} \rangle_{2^b}^p[j, 0]$
end
for $k = 0$ **to** $w - 2$ **do**
  for $j = 0$ **to** $n - 1$ **do**
    $\langle \mathbf{o} \rangle_{2^b}^p[j] = \langle \mathbf{o} \rangle_{2^b}^p[j] + \langle \mathbf{I} \rangle_{2^b}^p[j, k]$
  end
end
$\langle \mathbf{o} \rangle_{2^b}^p[j] = \mathcal{F}_{\mathsf{DIV}}(\langle \mathbf{o} \rangle_{2^b}^p, w)$

Algorithm 6 shows the protocol for average pooling from CrypTFlow2 that is employed in SecONNds. Here, $\mathcal{F}_{\mathsf{DIV}}$ is the protocol for integer division with a public divisor from CrypTFlow2. This field division protocol takes as input secret shares of the dividend and returns secret shares of the quotient.