SecONNds: Secure Outsourced Neural Network Inference on ImageNet

***Abstract*—The widespread adoption of outsourced neural network inference for tasks like image classification presents significant privacy challenges, as sensitive user data is processed on untrusted remote servers. Existing secure inference frameworks for large-scale datasets such as ImageNet often suffer from high computational overhead and communication costs, rendering them impractical for real-world deployment. We introduce SecONNds, a non-intrusive secure inference framework optimized for outsourced image classification on ImageNet. SecONNds integrates a novel fully Boolean Goldreich-Micali-Wigderson (GMW) protocol for secure comparison – addressing Yao's millionaires' problem – using preprocessed Beaver's bit triples generated from Silent Random Oblivious Transfer. Our novel protocol achieves upto 17-fold online speedup in nonlinear operations compared to state-of-the-art solutions while reducing communication overhead. To further enhance performance, SecONNds employs Number Theoretic Transform (NTT) preprocessing and leverages GPU acceleration for homomorphic encryption operations, resulting in speedups of 1.6× on CPU and 2.2× on GPU for linear operations. We also present SecONNds-P, a bit-exact variant that ensures full-precision, verifiable results in secure computation, matching the results of plaintext computations. Evaluated on a 37-bit quantized SqueezeNet model, SecONNds achieves an end-to-end inference time of 2.8 seconds on GPU and 3.6 seconds on CPU, with a total communication of 420 MiB. SecONNds' efficiency and reduced computational load make it well-suited for deploying privacy-sensitive applications in resource-constrained environments. The SecONNds framework is completely open source: https://github.com/SecONNds/SecONNds_SP25.**

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, such as image recognition, natural language processing, and recommendation systems. To handle the computational demands of these models, especially large ones like Deep Neural Networks (DNNs), it is common practice to outsource computations to remote cloud servers. This approach allows resource-constrained devices, such as mobile and embedded systems, to leverage powerful models by offloading the heavy computation and receiving the final result. However, this practice presents significant privacy risks, as sensitive user data is processed on remote servers that may not be fully trusted.

In response to these privacy concerns, the security and privacy research community has introduced *Secure Inference* frameworks, illustrated in Figure 1. Using cryptographic techniques, these frameworks ensure the protection of all private data from all parties involved, including the user's input and output of the inference process, and the proprietary model parameters of the service provider. These solutions allow users to utilize pre-trained NNs for inference without ever exposing their raw input data.

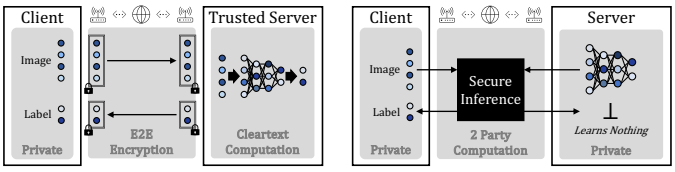

Fig. 1. Orthodox Outsourced Inference (left) vs. Secure Inference (right)

Neural networks are diverse in architecture but are essentially composed of an alternating series of multidimensional *linear* operations, and *nonlinear* operations which 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 [1] with models growing larger and larger in terms of parameters.

For efficient and practical secure inference, it is crucial to design computation algorithms for low-level neural network operations with efficient cryptographic primitives that provide privacy, considering the resource allocation of both parties. Employing a zero-trust model for secure inference demands that the framework does not reveal any private data and involves only the user and the service provider, necessitating a purely 2-party security model. This necessitates cryptographic primitives for *Secure 2-Party Computation* (2PC) protocols: Oblivious Transfer (OT)[2], Garbled Circuit (GC)[3], Goldreich-Micali-Wigderson (GMW)[4], and Homomorphic Encryption (HE) [5].

OT allows for secure exchange of messages and enables secure Look-Up Table (LUT) evaluations. Similar to how FPGA boards are programmed, OT is universal in its expressiveness and serves as a foundation for GC and GMW. GC is a one-round protocol for securely evaluating Boolean circuits and was the first solution proposed for 2PC. GMW enables secure evaluation of arithmetic circuits on fixed-point data on top of boolean circuits, but it is a highly 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 structure in the encrypted data, allowing computations on ciphertexts that reflect as simple operations on the underlying plaintexts. While partial HE schemes [6], [7] offer a limited set of operations, fully HE schemes [5], [8], [9] enable arbitrary computations but are very slow.

Leveled-HE schemes [10], [11], [12], [13] strike a good balance by supporting arithmetic circuits ($+$, $\times$) on fixed-point data up to a fixed size (multiplicative depth) with usable performance. The main advantage of HE is in securely outsourcing computation; the user sends encryptions to the server, which performs all computations and returns the encrypted result 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.

2PC primitives (GC, GMW, HE) are tailored for fixed-point circuits, so it is common to convert neural network operations to fixed-point representations. This task has been thoroughly researched in machine learning through Quantized Neural Networks (QNNs) [14]. QNNs use fixed-point representations for all data, including the trained model parameters.

Almost all of the trained parameters of a model belong to its linear layers, and are only used for computing linear combinations of the inputs. Therefore, A model's size directly corresponds to the number of linear operations involved. With GC or GMW, the communication footprint of linear layers scales with the number of total scalar multiplications. In contrast, with HE, the communication footprint scales only with the sizes of the input and output vectors. Today, leveled-HE outperforms any other 2PC primitive in evaluating high-dimensional linear algebra operations on encrypted data [15].

Nonlinear operations comprise the remainder of the computation and are indispensable in neural networks. These nonlinear operations involve comparisons, and in fact, this was one of the first problems studied in secure computation by Andrew Yao [3], who dubbed it the Millionaires' Problem, $\mathsf{MILL} = \mathbb{1}\{i_0 > i_1\}$, which indicates if party-0's input (i_0) is greater than party-1's input (i_1). The bulk of online runtime of these operations in state-of-the-art secure inference protocols comes from the secure comparisons (Millionaires' protocol).

Solving the Millionaires' Problem involves securely evaluating a boolean circuit which is only supported by GC and GMW. GC involves a ciphertext expansion of $O(\lambda b)$[1] for comparing a pair of b bit secrets. A GMW protocol initialized with Silent OT Extension [16] performs the same task with an overhead of only $O(b)$, greatly propelling its performance over GC. However, existing GMW protocols still incur high computational overhead, especially for large-scale datasets like ImageNet, making them impractical for real-world applications.

Our Approach: We present **SecONNds**, a non-intrusive secure inference framework that develops new algorithms for the Millionaires' Problem within the GMW protocol and enhances HE-based linear operations using Number Theoretic Transform preprocessing and GPU acceleration. By holistically improving both nonlinear comparisons and linear computations, SecONNds significantly boosts the performance of secure inference, making it practical for large-scale applications like ImageNet. Our main contributions are as follows:

[1] λ is the computational security parameter, typically 128

+ New solutions to the Millionaires' Problem: We present $\mathcal{F}_{\mathsf{MILL}}$ for secure comparisons, 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, and an alternate variant that incurs only a logarithmic number of rounds, with slightly higher computation and communication costs. Both are further enhanced with an offline triple buffer and a chunked generator optimized for silent OT.

+ Efficient protocols for neural network operations: We develop new 2PC protocols for ReLU, Max Pooling, and Truncation using $\mathcal{F}_{\mathsf{MILL}}$ with offline triples. For linear algebra operations, we employ the BFV HE scheme [11], [12] featuring one-time preprocessing with Number Theoretic Transform (NTT) [15] and server-side GPU acceleration [19]. Our protocols achieve online speedups of $17\times$ for Max Pooling, $11\times$ for ReLU, $2.1\times$ for Truncation, and $2.2\times$ for convolution with HE on GPU, over prior art.

+ End-to-end implementation and evaluation: We evaluate our open-source implementation on a 37 bit quantized SqueezeNet model [20] for ImageNet [21]. SecONNds achieves an E2E runtime of just 2.8 seconds on GPU and 3.6 seconds on CPU for secure inference, involving 420 MiB of total communication. Its E2E performance is $4.2\times$ faster online compared to the state-of-the-art. We also implement SecONNds-P, a bit-exact variant that ensures full-precision, verifiable results in secure computation compared to plaintext.

SecONNds is ideal for enabling privacy-sensitive ML applications in resource-constrained environments. It allows service providers to securely outsource NN computation, delivering practical performance with robust security and reduced computational load for users. Moreover, being a non-intrusive framework, it offers foundational modules that apply to any neural network, regardless of the media type, without requiring any model finetuning, and cuts any reliance on training data. SecONNds is highly compatible – it is fully open-source, modular and dynamic, allowing for mixing between different preprocesses and protocol optimizations at runtime, e.g., linear layers with online/offline/no NTT preprocessing, millionaires' protocol with log round complexity, refreshing the buffer, etc.

II. Background

A. Mathematical Notation

In this section, we introduce the mathematical notation used throughout this paper. Integer vector fields are denoted by $\mathbb{Z}_Q^N$ where N is the dimensionality of the vector space and Q 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 $\mathbb{Z}_Q$. Here N is the polynomial degree modulus and Q is the coefficient modulus.

Scalars in $\mathbb{Z}_N$ are denoted by 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 by bold lowercase letters with an overline $\overline{\mathbf{p}}$. A HE plaintext encoding a secret $\mathbf{m}$ is denoted as $\overline{\mathbf{pt}}_{\mathbf{m}}$ 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 (OR) and multiplication (AND) in $\mathbb{Z}_2$. Party $\mathcal{P}_p$'s linear secret share of i over $\mathbb{Z}_N$ is denoted by $\langle i \rangle_p^N$, therefore $i = \langle i \rangle_p^N + \langle i \rangle_p^N \bmod N$. The indicator function is denoted by $\mathbb{1}\{\textit{condition}\}$, it is 1 if the *condition* is satisfied, and 0 otherwise.

B. Secure 2-Party Computation (2PC)

Secure 2-party computation enables two parties to jointly compute a function over their private inputs while keeping them confidential. Various cryptographic primitives facilitate 2PC, each with trade-offs in terms of efficiency, communication overhead, and computational complexity.

Oblivious Transfer (OT) [2] is a cryptographic primitive that allows a sender to transfer one out of many pieces of information to a receiver without knowing which piece was transferred, and ensuring nothing is learnt about the other pieces. OT has various variants based on functionality.

Correlated OT (COT): In a 2 Correlated OT, denoted by $\binom{2}{1}$–COT_b, the sender inputs a b bit string m, and the receiver obtains a b bit string $m_c = m + c \cdot \delta$, where δ is a fixed b bit correlation known to the sender, and $c \in 0, 1$ is the receiver's choice bit. The outputs are correlated according to the sender's input δ. COT is a foundation protocol, commonly used for generating correlated randomness between parties.

Random OT (ROT): Random OT is a variant where the sender's messages are randomly generated, and the receiver obtains one of them based on their choice bit. In a $\binom{2}{1}$–ROT_b, the sender obtains two random b bit strings $\{r_0, r_1\}$, and the receiver obtains one string $r_c \in \{r_0, r_1\}$, where $c \in 0, 1$ is the receiver's choice bit. ROT is valuable for generating shared randomness without inputs from either party

Chosen OT: In standard OT (also called Chosen OT), the sender has two messages (m_0, m_1), and the receiver obtains m_c, where c is their choice bit, without learning anything about m_{1-c}. The sender remains oblivious to the receiver's choice c. Chosen OT is essential when one party needs to send specific messages to the other based on the receiver's choice. For example, to evaluate a secure LUT, the sender sets the entries with its messages, and the receiver indexes with its private choice to learn only the corresponding entry (message).

On the algorithm side, efficient OT extension protocols [22], [16] allow a large number of OTs to be generated from a small number of base OTs, significantly improving efficiency. We include an extended discussion of OT extension in Appendix A.

Garbled Circuits (GC) [3] is a 2PC protocol in which one party (the garbler) anonymizes and encrypts a Boolean circuit pertaining to the computation, such that the other party (the evaluator) can evaluate it on its input without learning any intermediate values or other inputs. Each wire in the circuit is assigned two λ bit random keys (labels), one for logical 0 and the other forms 1. The garbler encrypts the truth table of each gate so that the evaluator can compute the output keys given the input keys without learning the underlying logical values.

Correlated OT is used to transmit the evaluator's input wire labels without revealing the input bits to the garbler. The garbler prepares correlated wire labels, and through $\binom{2}{1}$–COT_λ, the evaluator obtains the correct wire labels corresponding to their input bits. This ensures that the garbler remains oblivious to the evaluator's inputs while the evaluator can proceed with the circuit evaluation.

While GC provides security with a single communication round, it can incur high computational and communication costs due to the need to encrypt every gate in the circuit and handle large garbled tables where each bit is represented with a large λ bit ciphertext.

Goldreich-Micali-Wigderson (GMW) [4] is an interactive protocol based on Linear Secret Sharing Schemes (LSSS) [23]. Parties, $\mathcal{P}_p$ for $p \in \{0\colon \text{Server}, 1\colon \text{Client}\}$, hold a secret share $\langle i \rangle_p^N$ for every private input $i \in [0, N)$, such that $\langle i \rangle_p^N + \langle i \rangle_p^N \bmod N = i$. For linear operations like addition, computation can be performed locally without interaction. However, secure multiplication (AND gates) requires interaction between the parties. During the evaluation of an AND ($\wedge$) gate, the parties engage in OT protocols to obtain shares of the product without revealing their private inputs.

Similar to GC, GMW also involves 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. When it comes to computation, while both GC and GMW 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. The round complexity of GMW can be significantly reduced with circuit randomization and correlated pseudorandomness using Beaver's bit triples.

Beaver's Bit Triples [24], [17] enable fast computation of an AND gate without relying on Chosen OT during the online phase. A Beaver's bit triple consists of $\left\{\langle a \rangle_p^2, \langle b \rangle_p^2, \langle c \rangle_p^2\right\}$, where a and b are random bits, and $c = a \wedge b$. This can be generated offline using 2 calls to $\binom{2}{1}$–ROT_1. During the online phase, given secret-shared inputs $\langle x \rangle_p^2$ and $\langle y \rangle_p^2$, the parties compute local corrections with the pre-shared triple values and exchange these correction bits to adjust their output shares (illustrated in Algorithm 4 of Appendix C).

Homomorphic Encryption (HE) allows clients to encrypt data, send it to a server for computation, and then decrypt the results themselves. This reduces communication overhead, as only input and output data are transferred, while the server handles most of the computation. Fully Homomorphic Encryption (FHE), such as Gentry's scheme [5], TFHE [9], and FHEW [8], supports arbitrary computations through bootstrapping but adds computational costs. Leveled HE limits computations but improves efficiency at the cost of increased ciphertext size. Many schemes, like BGV [10], BFV [11], [12], and CKKS [13], rely on *Ring Learning With Errors* (RLWE) [25], involving small-error polynomial equations in rings $\mathcal{R}_Q^N$.

HE supports addition, multiplication, and rotation operations on ciphertexts. Multiplications have the highest noise growth, affecting overall error, but they can be optimized for convolution computations using the Number Theoretic Transform (NTT), reducing complexity to $O(N \log N)$. In contrast, rotations are computationally slow and require key-switching, which involves additional special keys $\overline{\mathbf{ek}}$ and further increases noise. Error growth impacts the efficiency and security of HE, especially depending on modulus Q, making strategic parameter selection essential. We discuss more about RLWE-HE and the role of NTT in Appendix F.

2PC Cost Comparison. Table I highlights the costs of different secure two-party computation protocols – GC, GMW with Beaver's triples, and HE – for performing a secure matrix-vector product of size $N \times N$. In this setting, the client holds a private input vector, and the server holds a private input matrix (e.g., trained weights in a Support Vector Machine model). The goal is for the client to obtain the output vector without revealing its input or learning the server's matrix.

GC requires the server to garble the circuit locally during the offline phase, incurring computational costs. There is no communication required in the offline phase. In the online phase, the server sends the garbled circuit to the client, resulting in a communication cost proportional to λN^2, where λ is the security parameter. Additionally, N Correlated OTs are used to share the input wire labels corresponding to the client's input vector during the online phase. The client then evaluates the garbled circuit using its input labels and learns the output.

GMW involves generating N^2 Beaver's triples using $2N^2$ Random OTs during the offline phase, 1 triple per scalar multiplication required in the matrix-vector product. Both parties incur symmetric computational and communication costs during this phase. In the online phase, the parties exchange correction bits in a single round of communication, leading to a communication cost proportional to N^2 and maintaining symmetric computation between both parties.

HE, during the offline phase, may involve the server preprocessing its matrix with NTT [15]. In the online phase, the client encrypts its input vector and sends it to the server. The server performs the matrix-vector multiplication homomorphically using its private matrix and returns the encrypted output vector to the client. The communication cost is proportional to N, and the computational burden is primarily on the server, which performs the homomorphic operations locally.

TABLE I
2PC PERFORMANCE FOR $N \times N$ MATRIX-VECTOR PRODUCT

2PC Protocol	Offline Rounds	Offline Comm.	Offline Comp.	Online Rounds	Online Comm.	Online Comp.
GC	0	0	Server	1	$O(\lambda N^2)$	Client
GMW	1	$O(N^2)$	Both	1	$O(N^2)$	Both
HE	0	0	Server	1	$O(N)$	Server

C. Secure Inference Frameworks

Many major tech companies, such as Apple [26], Google [27], Microsoft [28], and Amazon [29], are making a huge push into privacy-focused solutions for AI inference, but they choose to use Trusted Execution Environments (TEEs). For example, Apple's Private Cloud Compute (PCC) uses a TEE built on custom Apple silicon to protect user data during cloud-based AI processing. PCC ensures data is processed securely and in a stateless manner, reducing risks associated with data retention. The main issue with such outsourced inference systems (depicted in the left half of Figure 1) is they protect the data with encryption only during transit, the computation is still performed in cleartext, albeit inaccessible due to hardware privileges. Despite their secure design, limitations inherent in TEEs, like potential side-channel vulnerabilities, hamper its applicability to privacy critical applications [30], [31], [32].

In contrast, cryptographic methods for secure inference (depicted in the right half of Figure 1), involving 2PC, do not depend on any hardware components and instead leverage strong cryptographic guarantees to protect data in all phases (locations): rest (memory), transit (network) and computation (processor). These frameworks enable neural network computations on private data, without revealing any sensitive information to any participant. Over the years, several frameworks have been developed, leveraging various cryptographic techniques to balance efficiency and privacy.

CryptoNets [33] is the first secure inference framework to run a Convolution Neural Network on MNIST [34]. 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 [35] is the first non-intrusive[2], mixed-protocol framework to perform inference on CIFAR-10 [36]. 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) convolution, and smooth functions are approximated with splines (piecewise polynomials). MiniONN features an offline preprocessing phase to set up beaver's bit triples [17] using HE. During the online phase, which involves evaluating the inference result of the private image, they use GMW with preprocessed triples for all linear operations and GC for comparisons. 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 later frameworks, strictly targeting E2E performance, adhere to this Mixed-protocol design.

Gazelle [37] 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.

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

CrypTFlow2 [38] introduced a new protocol for the millionaires' problem leveraging the state-of-the-art optimizations and techniques for IKNP-style OT extension [39], [40]. 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 [21]. It employs a *faithful truncation* scheme that ensures bit-exact results in secure computation compared to plaintext.

Cheetah [41] 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 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).

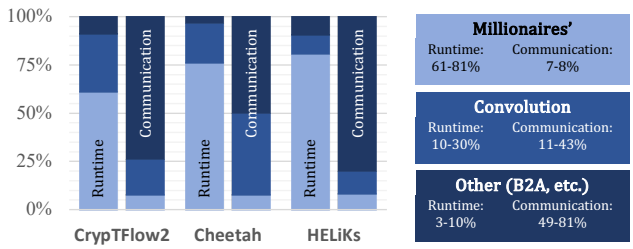

Fig. 2. Secure Inference costs per protocol: Millionaires', Convolution, Other

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.

Transformer Inference. Recently, there has also been a huge push in research for secure inference on transformer models with frameworks like, SIRNN [42], Iron [43], BOLT [44], and BumbleBee [45]. 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.

D. Computational Bottlenecks

Despite advancements, secure inference frameworks face significant computational challenges, particularly in non-linear operations involving secure comparisons. The Millionaires' Protocol, which determines if one private value is greater than another, is essential for activation functions like ReLU and operations like Max Pooling. To quantify the overhead, we evaluated the single-threaded runtime and communication costs of CrypTFlow2, Cheetah, and HELiKs per protocol for running a 37 bit SqueezeNet [20] model under the same compute and network setup (described in Section VIII). We used the same Silent OT Extension [18] back-end for all frameworks. The graph illustrated in Figure 2, shows the distribution of total runtime and communication across each protocol: Millionaires' (secure comparisons), Convolution (with HE), and Other.

Millionaires' Protocol is the most significant contributor to runtime across all frameworks, accounting for $61-76\%$ of the total. CrypTFlow2 and HELiKs with faithful truncation spend 140 seconds for secure comparisons, while Cheetah reduces this to around 60 seconds by employing optimizations like 1 bit approximate truncation and delayed computation. However, even with these improvements, secure comparisons remain the primary bottleneck due to their reliance on bit-level operations and multiple rounds of interaction. In terms of communication, the cost from comparisons is very low $7-8\%$, with Cheetah requiring less than half of CrypTFlow2 and HELiKs.

Convolution Operations are the next major bottleneck, with runtime contributions ranging between 17% and 30%.

Cheetah brings the convolution time down to 17 seconds, compared to 70 seconds for CrypTFlow2 and 34 seconds for HELiKs. Its performance improves significantly with the elimination of HE rotations and could further benefit from the optimizations introduced by HELiKs like Number Theoretic Transform (NTT) pre-computation. Convolution operations also contribute to $11-43\%$ of the total communication. HELiKs, with a communication cost of 124 MiB, outperforms both CrypTFlow2 (217 MiB) and Cheetah (205 MiB), owing to effective noise management during computation that results in smaller ciphertext sizes.

Other Operations, including local plaintext operations and binary-to-arithmetic share conversions (B2A), contribute modestly to the runtime across all frameworks but impose significant communication overhead. The B2A operations have large communication costs since they involve translating 1 bit inputs to b bit field elements.

Implications for Transformers. The computational challenges posed by secure comparisons extend beyond basic CNN operations to more complex architectures like transformers. Recent transformer secure inference frameworks heavily rely on the millionaires' protocol for range checks in activation functions. Frameworks like SIRNN [42] and BOLT [44] directly employ CrypTFlow2's protocol with IKNP-style OT extensions, while Iron [43] and BumbleBee [45] use Cheetah's variant with silent-OT extension. While these frameworks introduce novel methods for approximating non-linear operations with splines – primarily focusing on reducing the number of polynomial segments and consequently the calls to millionaires' protocol – they maintain the fundamental comparison protocol unchanged. This widespread reliance on existing comparison protocols, coupled with their significant performance overhead shown in our analysis, highlights a critical need for new approaches to secure comparisons in privacy-preserving neural network inference.

III. Threat Model

SecONNds operates under a two-party semi-honest (honest-but-curious) security model involving a client with private input data and a server with a private model. The framework assumes both parties execute protocols correctly but may record all observed values, no collusion between parties exists, and network adversaries can observe but not modify communications. Under this security model, the framework ensures: (1) *Client Privacy* – the server learns nothing about the client's input image or inference result, (2) *Model Privacy* – the client learns nothing about the model parameters beyond what can be inferred from the output label, and (3) *Computational Security* with 128 bit security parameter based on standard cryptographic assumptions. The model excludes active adversaries who may deviate from protocol specifications.

IV. Millionaires' Protocol

The Millionaires' problem was conceptualized by Yao [3] 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 [38]. The core part of this algorithm comes from the work of Garay et al. [46], who proposed a novel approach of decomposing the two large b bit inputs, x of one party ($\mathcal{P}_0$) and y of the other party ($\mathcal{P}_1$), into $q = b/m$ consecutive m bit segments each, $\{x_0, ..., x_{q-1}\}, \{y_0, ..., y_{q-1}\}$, and computing the result with an arithmetic circuit over m bit numbers. This algorithm is illustrated in Figure 3 for 8 bit inputs ($b = 8$) with a segment size of $m = 2$ bits.

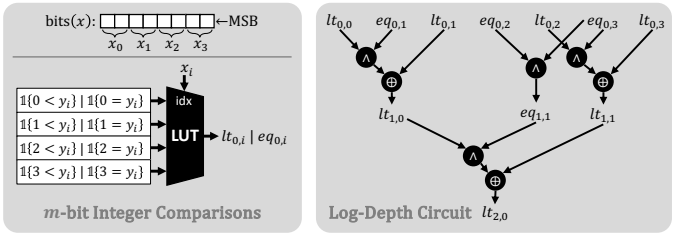

Fig. 3. Millionaires' Protocol with secure LUT (Chosen OT) [46], [38], [41]

The computation begins with evaluating the inequality $lt_{0,i} = \mathbb{1}\{x_i < y_i\}$, and equality $eq_{0,i} = \mathbb{1}\{x_i = y_i\}$ results for all segment-pairs $i \in [0, q)$. This is followed by an 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:

$$lt_{j,i} = lt_{j-1,2i} + eq_{j-1,2i} \times lt_{j-1,2i+1} \quad (1)$$

$$eq_{j,i} = eq_{j-1,2i} \times eq_{j-1,2i+1} \quad (2)$$

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. [46] 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. [47] showed how integer comparisons could be evaluated using OT.

CrypTFlow2's algorithm mixes the OT based integer comparisons of Blake et al. [47] with the log-depth arithmetic circuit of Garay et al. [46] for combining the results of the integer comparisons. It improves 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, a boolean version of the log-depth circuit of Garay et al. [46] is implemented by replacing $+$ with bit-XOR ($\oplus$) and $\times$ with bit-AND ($\wedge$). The $\wedge$–gates in this boolean circuit are evaluated with a GMW protocol $\mathcal{F}_{\mathsf{AND}}$ using bit triples. It optimizes 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 ($eq_{0,3}$ in

Figure 3). 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 $\mathcal{F}_{\mathsf{AND}}$).

Cheetah optimized this by porting the OT primitives to silent OT extension [18] which 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. It follows the strategy presented by Asharov et al. [40] and generates a bit triple with 2 calls to $\binom{2}{1}$–ROT_1. Since these ROTs use silent OT extension, $\binom{2}{1}$–ROT_1 is almost free in terms of communication. Cheetah follows CrypTFlow2 on setting the segment size $m{=}4$ and using $\binom{16}{1}$–OT_2 for the best performance. For $m{=}4$, communication requires $9b$ bits for integer comparisons and $2b + 4 - 4\lceil \log b \rceil$ bits for the boolean circuit, totaling approximately $11b$ bits, while $m{=}1$ requires $5b$ bits for comparisons and $8b - 4 - 4\lceil \log b \rceil$ bits for circuit evaluation, totaling about $13b$ bits (an 18% increase). In our evaluation with 2^{13} comparisons of 32 bit numbers, while $\binom{2^m}{1}$–OT_2 operations take 90 ms for both settings, $\binom{2}{1}$–ROT_1 operations vary drastically: 110 ms for $m{=}1$; 15 ms for $m{=}4$.

A. Fully-Boolean Algorithm

SecONNds features a fully-boolean GMW protocol $\mathcal{F}_{\mathsf{MILL}}$ shown in Algorithm 1, that eliminates LUT evaluations (chosen OTs). For segments of size $m = 1$, while prior works employ heavy $\binom{2}{1}$–OT_2 for the simple task of comparing a pair of bits, SecONNds builds on the observation 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 $\langle eq_{0,i} \rangle_0^2 = 1 \oplus x_i$ and $\mathcal{P}_1$ sets its share $\langle eq_{0,i} \rangle_1^2 = y_i$, 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 triple generation) and takes less than 40 ms for 2^{13} comparisons with 32 bit numbers. It requires a communication of just 4 bits per $\mathcal{F}_{\mathsf{AND}}$ and a total of $4b$ bits.

For the log-depth circuit, 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:

$$lt_i = lt_{i-1} \oplus lt_{0,i} \oplus (eq_{0,i} \wedge lt_{0,i-1}) \text{ for } i \in [1, b) \quad (3)$$

Following this linear strategy with $lt_0 = 0$, 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 $m = 1$. This strategy incurs a communication cost of only $4(b-1)$ and brings the total communication footprint of the millionaires' protocol to under $8b$ bits which is 27% lower than the $11b$ bits cost of the $\binom{16}{1}$–OT_2 with $m = 4$.

Algorithm 1: $\mathcal{F}_{\mathsf{MILL}}$ Millionaires' in **SecONNds**

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

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

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

- /* Bit Extraction & Share Generation */
- $\langle b_0 \rangle_p^2 = \Big(\big(i_p / 2^i \bmod 2 \big) \oplus g' \Big) \wedge p'$
- $\langle b_1 \rangle_p^2 = \Big(\big(i_p / 2^i \bmod 2 \big) \oplus g \Big) \wedge p$
- /* Bit Equality & Inequality Comparisons */
- $\langle \mathbf{b_{eql}} \rangle_p^2[i] = \langle b_0 \rangle_p^2 \oplus \langle b_1 \rangle_p^2$
- $\langle \mathbf{b_{l/g}} \rangle_p^2[i] = \mathcal{F}_{\mathsf{AND}} \Big(\langle b_0 \rangle_p^2, \langle b_1 \rangle_p^2 \Big)$

/* Combining Bit Results */

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

- $\langle b_{and} \rangle_p^2 = \mathcal{F}_{\mathsf{AND}} \Big(\langle \mathbf{b_{eql}} \rangle_p^2[i+1], \langle \mathbf{b_{l/g}} \rangle_p^2[i] \Big)$
- $\langle \mathbf{b_{l/g}} \rangle_p^2[i+1] = \langle \mathbf{b_{l/g}} \rangle_p^2[i+1] \oplus \langle b_{and} \rangle_p^2$

$\langle o \rangle_p^2 = \langle \mathbf{b_{l/g}} \rangle_p^2[b-1]$

TABLE II
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**

The linear approach 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 II, 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 $\binom{16}{1}$–OT_2 version with $m = 4$. 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. In the following section, we show how to significantly lower the online runtime from 75 ms to under 5 ms with offline triple generation.

B. Chuncked Triple Generation

To efficiently generate bit triples with 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 crypto-Tools library [48]. 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 $O\left(n^2\right)$, 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 ReLU and Truncation operations involved in quantized Convolutional Neural Networks (CNNs). We detail the protocols for Max Pooling and Average Pooling in Appendix D and E respectively.

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 dReLU is 1 and secret shares of 0 if dReLU 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 2: $\mathcal{F}_{\mathsf{ReLU}}$ ReLU

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

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

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

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

3 $i_{mill} = (-1)^p\left|\langle i\rangle_p^{2^b}\right| + p\cdot\left(2^{b-1}-1\right)$

4 $\langle w\rangle_p^2 = \mathcal{F}_{\mathsf{MILL}}\left(b-1, 1, i_{mill}\right)$

5 $\langle i_{drelu}\rangle_p^2 = \mathsf{MSB}\left(\langle i\rangle_p^{2^b}\right) \oplus \langle w\rangle_p^2 \oplus p'$ // dReLU result

6 $\delta = \left(1 - 2\cdot\langle i_{drelu}\rangle_p^2\right)\cdot\langle i\rangle_p^{2^b}$ // Delta for COT

7 $c = \langle i_{drelu}\rangle_p^2$ // Choice for COT

8 $m_s = \binom{2}{1}$–COT_b–$\mathsf{send}(\delta)$; $m_r = \binom{2}{1}$–COT_b–$\mathsf{receive}(c)$

9 $\langle o\rangle_p^{2^b} = \left(\langle i\rangle_p^{2^b}\cdot\langle i_{drelu}\rangle_p^2 + m_r - m_s\right) \bmod 2^b$

Here, the wrap bit w is securely computed using the millionaires' protocol described in Section IV. After computing dReLU, the protocol uses a secure multiplexer functionality, MUX, realized with two calls to $\binom{2}{1}$–COT_b. The dReLU result serves as the selection bit. If the dReLU result is 1, the MUX outputs new shares of the input; if the dReLU result is 0, the MUX outputs shares of zero.

B. 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 3. The core concept of this approach is to represent the data in its unsigned form as follows:

$$w = \mathbb{1}\left\{\langle i\rangle_0^{2^b} + \langle i\rangle_1^{2^b} > 2^b - 1\right\}$$
$$i = \langle i\rangle_0^{2^b} + \langle i\rangle_1^{2^b} - w\cdot 2^b$$
$$i/2^s \approx \langle i\rangle_0^{2^b}/2^s + \langle i\rangle_1^{2^b}/2^s - w\cdot 2^{b-s}$$

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 [49], [41] 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, $w = \mathsf{MSB}\left(\langle i\rangle_0^{2^b}\right) \wedge \mathsf{MSB}\left(\langle i\rangle_1^{2^b}\right)$ 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.

Algorithm 3: $\mathcal{F}_{\mathsf{Trunc}}$ Truncation

Input: Right shift amount s ;
Bit i_{msb} indicating if $\mathsf{MSB}(i)$ is known;
Input secret share $\langle i \rangle_p^{2^b}$ where $i < 2^{b-1}$

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

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

/* Wrap Bit Computation */

if i_{msb} **then**

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

else

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

/* Wrap B2A Share Conversion */

if $p = 0$ **then**

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

  $m_s = \binom{2}{1}\text{–}\mathsf{COT}_b\text{–}\mathsf{send}(\delta)$

  $\langle w \rangle_p^{2^b} = \langle w \rangle_p^2 - m_s$

else

  $c = \langle w \rangle_p^2$ // Choice for COT

  $m_r = \binom{2}{1}\text{–}\mathsf{COT}_b\text{–}\mathsf{receive}(c)$

  $\langle w \rangle_p^{2^b} = \langle w \rangle_p^2 + m_r$

/* Final Truncation Result */

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

C. 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 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 [50] 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

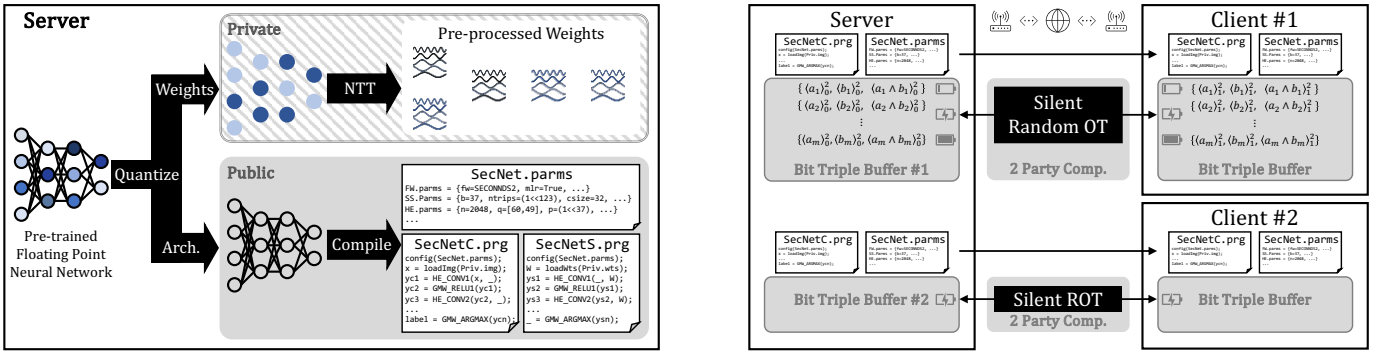

Fig. 4. Workflow of server (left) and client (right) setup in SecONNds.

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 [51] in CUDA [52]. 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. Framework Overview

SecONNds follows a modular design with distinct setup phases for server and client, illustrated in Figure 4. The server's setup involves three key steps: (1) quantizing the pretrained model to fixed-point representation, packing and encoding weights into HE plaintexts with NTT transformations for efficient polynomial multiplication, (2) compiling the network architecture into configuration files specifying protocol parameters including secret sharing bitwidth, $\mathcal{F}_{\mathsf{MILL}}$ variant (log-depth/linear), NTT preprocessing settings for convolutions, and triple buffer configurations, and (3) generating program files containing the complete computation graph and layer-wise execution order for both server (`SecNetS.prg`) and client (`SecNetC.prg`) implementations. This preprocessing phase is query-independent and needs to be performed only once unless the model parameters are updated.

A client requesting secure inference service first receives the model-specific configuration and program files. Based on these specifications, the client generates the necessary number of Beaver's bit triples using silent ROT, determined by the model architecture and planned number of inferences. The triple generation process, being input-independent, is entirely preprocessing and can be performed offline before the actual inference requests. These triples are stored in a buffer that automatically refills when exhausted during computation, with dynamic size adjustment capabilities to handle varying protocol requirements.

The online inference protocol executes layer-by-layer with both parties maintaining secret shares of intermediate activations throughout the network. Nonlinear operations (ReLU, Max Pooling) employ GMW protocol with the preprocessed triples, requiring interaction only for AND gates where parties exchange correction bits. For linear operations (Convolution, Fully Connected layers), the client encrypts and sends its activation shares to the server, which adds its own shares to these ciphertexts, performs linear operations using the NTT-preprocessed weights, applies a random mask for security, and returns the encrypted result to the client for decryption into output shares. This mixed-protocol approach optimally balances computation and communication overhead.

SecONNds ensures perfect security under the semi-honest model - the client learns only the final classification label while the server learns nothing about the input or intermediate values. Its modular design allows for runtime protocol selection and parameter configuration through the configuration files, enabling optimization for different network conditions and performance requirements. For example, developers can toggle between log-depth and linear variants of $\mathcal{F}_{\mathsf{MILL}}$ based on network latency, or enable/disable NTT preprocessing for convolutions depending on available computational resources.

VIII. Evaluation

SecONNds is implemented in the OpenCheetah [53] variant of the Secure and Correct Inference (SCI) Library [54]. We implement SecONNds for secret-sharing with $\mathbb{Z}_{2^b}$ and SecONNds-P for $\mathbb{Z}_P$, where P is a BFV-SIMD plaintext prime modulus. SecONNds uses 1-bit approximate truncation, while SecONNds-P employs faithful truncation and returns bit-exact results compared to the plaintext model. Both use the fast version of $\mathcal{F}_{\mathsf{MILL}}$ with the linear strategy by default. If the low-round variant is employed for a framework, it is denoted by LR in parentheses, e.g., SecONNds (LR).

For all our evaluations, we outfitted all frameworks being compared with: Ferret Silent OT Extension [18] from the EMP-OT Library [55]; BFV HE scheme [11], [12] from the SEAL Library [51]; and server-side GPU acceleration with the HE Evaluator from the Troy Library [19]. All CPU operations of both parties were 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 were performed on an *NVIDIA RTX A6000* system.

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

Nonlinear	CrypTFlow2		HELiKs		Cheetah		SecNN–P (LR)		SecONNds-P		SecNN (LR)		**SecONNds**	
Operation	Time	*Comm.*	Time	*Comm.*	Time	*Comm.*	Time	*Comm.*	Time	*Comm.*	Time	*Comm.*	Time	*Comm.*
Truncation	4.65	*307*	4.65	*307*	0.12	*4.86*	0.81	*284*	0.51	*267*	0.07	*4.79*	0.06	*4.62*
ReLU	6.73	*302*	6.73	*302*	2.53	*110*	1.72	*246*	0.87	*186*	0.72	*116*	0.23	*88.7*
Max Pooling	7.10	*398*	7.10	*398*	4.34	*154*	2.04	*326*	1.03	*247*	0.89	*151*	0.26	*117*
Avg Pooling	0.03	*4.19*	0.03	*4.23*	0.06	*4.43*	0.02	*4.20*	0.02	*4.19*	0.06	*4.42*	0.05	*4.45*
Arg Max	0.06	*0.21*	0.06	*0.19*	0.02	*0.11*	0.02	*0.19*	0.02	*0.20*	0.01	*0.11*	0.01	*0.11*

TABLE IV
OFFLINE TRIPLE GENERATION COSTS

Framework	**Triples**	**Runtime (s)**	**Comm. (MiB)**
SecONNds-P (LR)	1.2×10^9	27.16	60.41
SecONNds-P	8.2×10^8	22.43	43.12
SecONNds (LR)	4.9×10^8	10.61	26.36
SecONNds	3.4×10^8	8.31	17.01

TABLE V
ONLINE, OFFLINE RUNTIMES AND COMMUNICATION FOR CONVOLUTION

Framework	**Offline (s)**	**Online (s)**	**Comm. (MiB)**
CrypTFlow2	0.00	11.59	217.10
HELiKs	2.51	8.62	129.18
Cheetah	0.00	4.94	204.84
Cheetah (GPU)	0.00	3.72	204.84
SecONNds-P/(LR)	2.62	8.47	129.18
SecONNds/(LR)	0.76	3.09	204.84
SecONNds (GPU)	0.72	2.26	204.84

We evaluate the performance of SecONNds on the pre-trained SqueezeNet [20] CNN model from OpenCheetah [53]. The SqueezeNet model is uniformly quantized to fixed-point with a bitwidth of 32, achieving 79.6% Top-5 accuracy, the same as in prior works. SecONNds-P performs bit-exact computations and produces the same output logits as the cleartext model. SecONNds uses 1-bit approximate truncation; with just 0.0015% *Mean Absolute Percentage Error* (MAPE) in output logits, it bears no impact on accuracy. We also evaluate a ResNet50 model [56], which achieves 92.3% Top-5 accuracy with the same 37-bit setup and 12-bit scale and present the results in Appendix G.

A. Offline Preprocessing

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, no communication is required. For SqueezeNet, SecONNds requires 0.73 seconds for NTT preprocessing, while SecONNds-P requires 2.62 seconds with HE operations in $\mathbb{Z}_P$ from HELiKs. The server can reuse the NTT preprocessed weights over multiple queries unless the model parameters are updated or modified.

In addition, SecONNds also generates Beaver's triples offline for each query, which significantly improves the online performance of nonlinear operations. SecONNds-P requires $2.4\times$ more triples compared to SecONNds, due to the higher overhead associated with nonlinear operations in $\mathbb{Z}_P$. As mentioned in Section IV-A, the log-depth variant of the $\mathcal{F}_{\mathsf{MILL}}$ (LR) involves up to $2\times$ more AND ($\wedge$) gates. This leads to a larger volume of communication in each round, resulting in a total communication increase of approximately $1.5\times$ for all calls to $\mathcal{F}_{\mathsf{MILL}}$ in nonlinear operations, and longer runtimes by a factor of $1.6\times$, when compared the faster $\mathcal{F}_{\mathsf{MILL}}$ with the linear circuit.

B. Nonlinear layers

In Table III, we show the cumulative performance of the all nonlinear operations in a single secure inference on SqueezeNet: ReLU, Max Pooling, Average Pooling, ArgMax, and Truncation. SecONNds achieves substantial improvements over Cheetah in nonlinear operations, with $17\times$ faster Max Pooling, $11\times$ faster ReLU, and $2\times$ faster Truncation. Additionally, communication costs are reduced by up to 20%, with most gains observed in ReLU and Max Pooling.

Compared to CrypTFlow2 and HELiKs, SecONNds-P achieves significant runtime reductions – $8\times$ faster for ReLU, $7\times$ faster for Max Pooling, and $9\times$ faster for Truncation. Communication costs of SecONNds-P are lower by approximately 27% relative to these frameworks. The LR versions of SecONNds and SecONNds-P demonstrate increased runtimes and communication volumes due to the added $\wedge$-gates. Despite this, they still offer competitive performance, making them suitable when balancing latency and communication for different round complexities under different network conditions.

C. Linear Layers

The SqueezeNet model only consists of convolutions for linear layers, were evaluated for both CPU and GPU execution. In Table V, we show the performance of each framework for all convolutions in the model. SecONNds demonstrates reduced online runtime compared to Cheetah due to NTT preprocessing. On CPU, SecONNds achieves a runtime of 3.09 seconds, a $1.6\times$ improvement over Cheetah's 4.94 seconds. With GPU, SecONNds reduces runtime to 2.26 seconds achieving a speedup of $1.4\times$ over Cheetah (GPU). In terms of communication, SecONNds achieves the same performance as Cheetah for the convolution layers, with both requiring 204.84 MiB . However, HELiks and SecONNds-P show the best communication efficiency, requiring only 130 MiB, benefiting from improved noise management during HE operations.

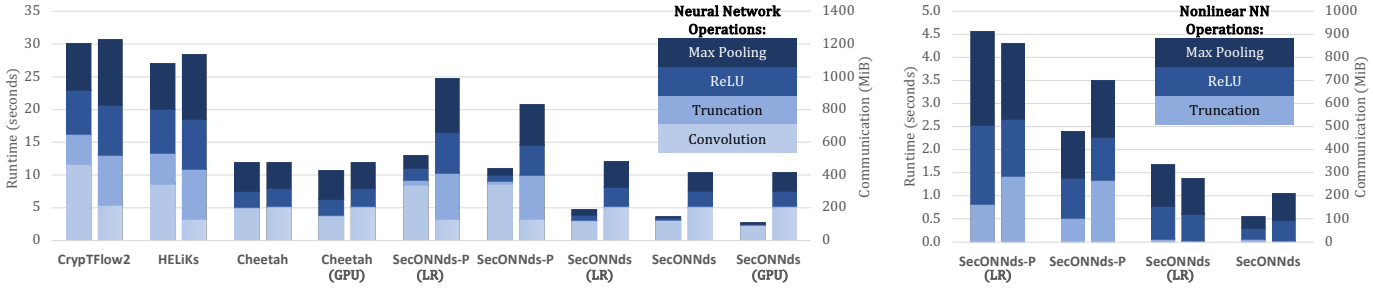

Fig. 5. End-to-end (E2E) runtime (left bar) and communication (right bar) performance of each framework (on horizontal axis) for the Neural Network (NN) operations in SqueezeNet, and a close up of nonlinear NN operations across the variants of SecONNds on the right.

D. E2E Evaluation

In the End-to-End (E2E) evaluations, shown in Figure 5, SecONNds demonstrates the best performance in terms of both runtime and communication efficiency, achieving total runtimes of 3.70 seconds on CPU and 2.87 seconds on GPU. This is a significant improvement over other frameworks, with a 3.24× speedup compared to Cheetah on CPU, which has an online runtime of 12 seconds. Even with GPU, SecONNds exhibits a speedup of approximately 3.8× compared to Cheetah's runtime of 10.79 seconds. For communication, SecONNds incurs a total of 420 MiB, showing a reduction of approximately 12% compared to Cheetah (478.93 MiB). SecONNds-P, which ensures bit-exact accuracy with full-precision truncation, demonstrates superior performance compared to HELiKs in both runtime and communication metrics. SecONNds-P achieves an online runtime of 11.06 seconds, which is a 2.46× speedup over HELiKs, which has an online runtime of 27.20 seconds. In terms of communication, SecONNds-P achieves a total of 834.17 MiB, a 27% reduction from 1141.95 MiB required by HELiKs.

For the logarithmic-depth (LR) variants, both SecONNds and SecONNds-P are designed to minimize the number of communication rounds at the cost of slightly increased computational and communication volume. Specifically, SecONNds (LR) achieves reduced communication rounds of 1084, while SecONNds-P (LR) operates with 1542 rounds. These figures represent a significant reduction compared to the non-LR versions of SecONNds and SecONNds-P, which require 4630 and 5800 rounds, respectively. Compared to Cheetah, which uses 900 rounds, the LR variants demonstrate competitive performance with a trade-off involving much higher computational effort and communication volume per round.

IX. Discussion

Limitations: SecONNds operates under the semi-honest security model, assuming honest protocol adherence, which restricts its use in malicious adversarial settings without incurring additional overhead. Additionally, secure inference still introduces considerable computational overhead compared to plaintext inference, which still poses challenges for real-time resource-constrained applications. Although the GPU library *Troy* [19] significantly accelerates elemental homomorphic encryption operations, the performance improvements for higher-level composite functions like convolutions are modest compared to highly optimized multithreaded CPU implementations. This highlights the need for specialized hardware acceleration targeting composite functions such as linear algebra operations to achieve comprehensive performance gains.

Alternate Approaches: In comparison to HE-only frameworks, which achieve secure inference with minimal interaction but suffer from substantial computational overhead due to FHE bootstrapping [5], SecONNds provides a more balanced solution by reducing these costs. Trusted Execution Environments (TEEs) like Intel SGX [57] offer low-latency inference within secure enclaves but rely on hardware trust assumptions and are vulnerable to side-channel attacks [58], [59]. SecONNds leverages a mixed-protocol strategy that optimizes both linear and nonlinear operations, achieving practical performance without depending solely on specialized hardware or compromising on security.

Future Directions: Future enhancements for SecONNds could include integrating neural network compression techniques such as pruning [60], knowledge distillation [61], or newer approaches to reduce model complexity and improve efficiency. Also, developing efficient protocols for the secure evaluation of complex nonlinear functions, like GeLU [62] and Swish [63], would enable support for a broader range of neural network architectures. Additionally, extending the security model to accommodate malicious adversaries and further optimizing hardware acceleration for composite functions could enhance both the security and performance of secure inference tasks.

X. Conclusion

SecONNds advances secure neural network inference by addressing key performance bottlenecks and demonstrating practical applicability on large-scale datasets. With a novel 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 under 3 seconds. Thus, by optimizing both linear and nonlinear operations and employing a mixed-protocol approach, SecONNds offers a balanced solution that provides strong security without prohibitive computational costs.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[2] 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

[3] A. C. Yao, "Protocols for secure computations," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, 1982, pp. 160–164. [Online]. Available: https://ieeexplore.ieee.org/document/4568388

[4] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, ser. STOC '87. New York, NY, USA: Association for Computing Machinery, 1987, p. 218–229. [Online]. Available: https://doi.org/10.1145/28395.28420

[5] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2009, aAI3382729. [Online]. Available: https://crypto.stanford.edu/craig/craig-thesis.pdf

[6] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On Data Banks and Privacy Homomorphisms," in *Foundations of Secure Computation*, R. A. DeMillo, D. P. Dobkin, A. K. Jones, and R. J. Lipton, Eds. Academic Press, 1978, pp. 165–179. [Online]. Available: https://luca-giuzzi.unibs.it/corsi/Support/papers-cryptography/RAD78.pdf

[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. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-48910-X_16

[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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-26954-8_16

[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] 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

[21] 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. [Online]. Available: https://ieeexplore.ieee.org/document/5206848

[22] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-45146-4_9

[23] 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

[24] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-46766-1_34

[25] 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

[26] Apple Inc., "Private Cloud Compute," accessed: 2024-10-25. [Online]. Available: https://security.apple.com/blog/private-cloud-compute/

[27] Google Cloud, "Confidential Computing," accessed: 2024-10-25. [Online]. Available: https://cloud.google.com/security/products/confidential-computing

[28] Microsoft Azure, "Confidential Compute," accessed: 2024-10-25. [Online]. Available: https://azure.microsoft.com/en-us/solutions/confidential-compute

[29] Amazon Web Services, "Nitro Enclaves," accessed: 2024-10-25. [Online]. Available: https://aws.amazon.com/ec2/nitro/nitro-enclaves/

[30] Z. KOU, S. Sinha, W. HE, and W. ZHANG, "Cache Side-channel Attacks and Defenses of the Sliding Window Algorithm in TEEs," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/10137116

[31] X. Zhang, J. Wang, Y. Cheng, Q. Li, K. Sun, Y. Zheng, N. Zhang, and X. Li, "Interface-Based Side Channel in TEE-Assisted Networked Services," *IEEE/ACM Transactions on Networking*, vol. 32, no. 1, pp. 613–626, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/10184979

[32] A. Javeed, C. Yilmaz, and E. Savas, "Microarchitectural Side-Channel Threats, Weaknesses and Mitigations: A Systematic Mapping Study," *IEEE Access*, vol. 11, pp. 48 945–48 976, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10123915

[33] 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

[34] 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. [Online]. Available: https://ieeexplore.ieee.org/document/6296535

[35] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious Neural Network Predictions via MiniONN Transformations," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 619–631. [Online]. Available: https://doi.org/10.1145/3133956.3134056

[36] 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

[37] 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

[38] 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

[39] 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

[40] 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

[41] 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

[42] 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

[43] 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. [Online]. Available: https://papers.neurips.cc/paper_files/paper/2022/file/64e2449d74f84e5b1a5c96ba7b3d308e-Paper-Conference.pdf

[44] 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

[45] 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

[46] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-71677-8_22

[47] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-540-30539-2_36

[48] P. Rindal, "cryptoTools," https://github.com/ladnir/cryptoTools, 2024, accessed: 2024-06-04.

[49] 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

[50] N. Markus, "Fusing batch normalization and convolution in runtime," 2023, accessed: 2024-07-10. [Online]. Available: https://nenadmarkus.com/p/fusing-batchnorm-and-conv/

[51] Microsoft, "Microsoft SEAL (Version 4.0)," https://github.com/microsoft/SEAL/tree/4.0.0, 2020, accessed: 2024-07-07.

[52] NVIDIA Corporation, "CUDA Toolkit," 2021, version 11.0. [Online]. Available: https://developer.nvidia.com/cuda-toolkit

[53] Alibaba Gemini Lab, "OpenCheetah," https://github.com/Alibaba-Gemini-Lab/OpenCheetah, 2024, accessed: 2024-07-07.

[54] MPC-MSRI, "Secure and Correct Inference (SCI) Library," https://github.com/mpc-msri/EzPC/tree/master/SCI, 2024, accessed: 2024-07-07.

[55] X. Wang and Others, "EMP-OT Library," https://github.com/emp-toolkit/emp-ot, 2024, accessed: 2024-07-07.

[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/He_Deep_Residual_Learning_CVPR_2016_paper.pdf

[57] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2487726.2488368

[58] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient Out-of-Order execution," in *27th USENIX Security Symposium (USENIX Security 18)*. Baltimore, MD: USENIX Association, Aug. 2018, p. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck

[59] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, "Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019, pp. 142–157. [Online]. Available: https://ieeexplore.ieee.org/document/8806740

[60] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 1135–1143. [Online]. Available: https://dl.acm.org/doi/10.5555/2969239.2969366

[61] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[62] D. Hendrycks and K. Gimpel, "Gaussian Error Linear Units (GELUs)," 2016. [Online]. Available: https://arxiv.org/abs/1606.08415

[63] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," 2018. [Online]. Available: https://openreview.net/forum?id=SkBYYyZRZ

[64] 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*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 669–684. [Online]. Available: https://doi.org/10.1145/2508859.2516668

[65] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-42045-0_15

[66] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, p. 792–807, Aug. 1986. [Online]. Available: https://doi.org/10.1145/6490.6503

[67] 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. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-48329-2_24

APPENDIX

A. Oblivious Transfer

Oblivious Transfer (OT), first introduced by Rabin [2], 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, a b bit value m, and doesn't learn anything, while the receiver only learns b bit $r + c \cdot \delta$. In $\binom{2}{1}$–ROT_b, the sender does not input anything and learns 2 random b bit values $\{r_0, r_1\}$ while the receiver learns only one of them, r_c for its input choice c.

In a variant called *random-choice* COT, the protocol automatically sets a random choice $c \in 0, 1$ and returns the corresponding random string m_c and c to the receiver. In another variant called *random-message* COT, the protocol automatically sets a random correlation δ and a random message r, and returns them to the sender. A $\binom{2}{1}$–ROT_b can be generated from a random-message $\binom{2}{1}$–COT_b by using a cryptographic hash function; sender hashes m, $m+\delta$ to obtain r_0, r_1 respectively and receiver hashes its ROT output m_c to get r_c. A $\binom{2}{1}$–OT_b can be generated from a $\binom{2}{1}$–ROT_b, the server uses r_0 and r_1 as one-time-pads to mask m_0 and m_1 and sends them to the receiver, the receiver uses r_c to unmask and learn m_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. [22] (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.

B. Silent OT Extension

The core components of Silent OT Extension are Punctured Pseudo-Random Function (PPRF) and Encoding with Learning Parity with Noise (LPN). 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* PRF (PPRF) [64], [65] can be evaluated on all points in the original PRF's domain except α, where it is punctured and returns 0 (zero).

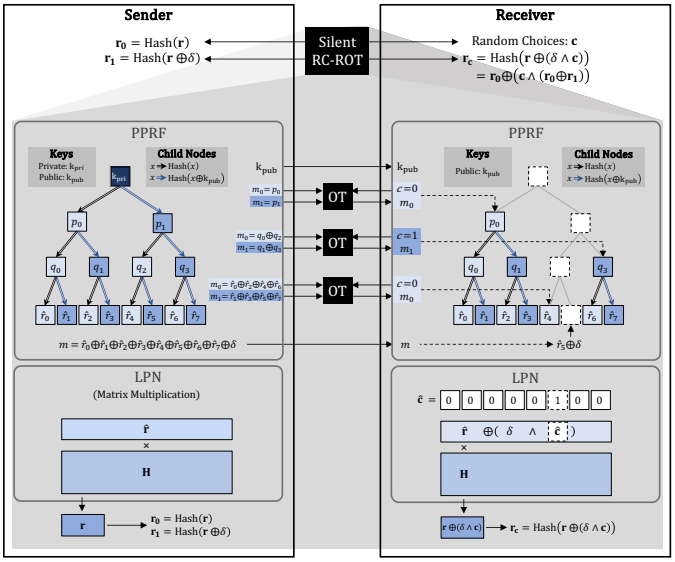

Fig. 6. High-level illustration of Silent ROT protocol.

A PPRF is generally constructed with a Goldreich, Goldwasser, and Micali PRF (GGM tree) [66] involving a tree expansion of $1 + \log n$ levels with one Hash per node for n OTs. Figure 6 illustrates the high-level operations involved in a silent Random-Choice ROT (RC-ROT). The PPRF generates $2n$ random-message COTs, with a choice vector $\hat{\mathbf{c}} = \mathbf{G} \times \mathbf{c} \in \mathbb{Z}_2^{2n}$ which is the sparse representation of $\mathbf{c}$ with the public generator matrix $\mathbf{G} \in \mathbb{Z}_2^{2n \times n}$ – the protocol assigns random values to $\mathbf{c}$.

The LPN (dual) assumption [67] states that $(\mathbf{H}, \mathbf{r} \cdot \mathbf{H}) \overset{c}{\approx} (\mathbf{H}, \mathbf{b})$: the product of the code matrix $\mathbf{H} = \mathbf{G}^{-1} \in \mathbb{Z}_2^{n \times 2n}$ and a random sparse vector $r \in \mathbb{Z}^2 n_2$ is computationally indistinguishable from a uniformly random vector $b \in \mathbb{Z}_2^n$. In Silent OT, LPN is used to compress $\hat{\mathbf{r}}$, $\hat{\mathbf{c}}$ and generate the COTs corresponding to the choice vector $\mathbf{c}$. This involves computing a matrix-vector product with a very large matrix – n is in the order of 10^8 for secure inference. To convert the COTs to ROTs, the sender computes two hashes of its output, while the receiver computes one hash to convert.

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 OTs, n.

C. Secure AND with Bit Triples

In secure multiparty computation, evaluating the logical AND ($\wedge$) operation securely is a fundamental primitive. One efficient method to achieve this is by using Beaver's bit triples. A Beaver's bit triple consists of shared random bits $\langle a \rangle_p^2, \langle b \rangle_p^2, \langle c \rangle_p^2$ for each party $\mathcal{P}_p$ ($p \in 0 : \text{Server}, 1 : \text{Client}$), such that $c = a \wedge b$ holds in the shared domain.

Algorithm 5: $\mathcal{F}_{\mathsf{MaxPool}}$ Max Pooling

Input: Flattened window size w
Input secret share array $\langle \mathbf{i} \rangle_p^{2^b}$ of size w
Output: output secret share $\langle o \rangle_p^{2^b}$

$\langle o \rangle_p^{2^b} = \langle \mathbf{i} \rangle_p^{2^b}[0]$
for $k = 1$ **to** $w - 1$ **do**
  $\langle o \rangle_p^{2^b} = \mathcal{F}_{\mathsf{ReLU}}\Big(\langle o \rangle_p^{2^b} - \langle \mathbf{i} \rangle_p^{2^b}[k]\Big) + \langle \mathbf{i} \rangle_p^{2^b}[k]$

Algorithm 4: $\mathcal{F}_{\mathsf{AND}}$ And with Beaver's Bit Triples

Input: Input secret shares $\langle x \rangle_p^2$ and $\langle y \rangle_p^2$
Output: Output secret share $\langle z \rangle_p^2$, s.t. $z = x \wedge y$

/* Triple Retrieval */
$\Big\{\langle a \rangle_p^2, \langle b \rangle_p^2, \langle c \rangle_p^2\Big\} = \mathsf{TripleGen.get}(1)$
/* Correction bit Computation */
$\langle e \rangle_p^2 = \langle a \rangle_p^2 \oplus \langle x \rangle_p^2$
$\langle f \rangle_p^2 = \langle b \rangle_p^2 \oplus \langle y \rangle_p^2$
/* Correction bit Reveal */
$\mathsf{Send}(\langle e \rangle_p^2)$; $\mathsf{Send}(\langle f \rangle_p^2)$
$\langle e \rangle_{p'}^2 = \mathsf{Receive}()$; $\langle f \rangle_{p'}^2 = \mathsf{Receive}()$
$e = \langle e \rangle_p^2 \oplus \langle e \rangle_{p'}^2$; $f = \langle f \rangle_p^2 \oplus \langle f \rangle_{p'}^2$
$\langle z \rangle_p^2 = (p' \wedge e \wedge f) \oplus (e \wedge \langle b \rangle_p^2) \oplus (f \wedge \langle a \rangle_p^2) \oplus \langle c \rangle_p^2$

A bit triple can be constructed using two Random-Choice ROTs (RC-ROT), where each is a $\binom{2}{1}$–ROT_1. The construction process is as follows:

1) **First RC-ROT Instance**:
   - The server acts as the receiver, obtaining a random choice bit $d \in 0, 1$ and the corresponding random message r_d.
   - The client acts as the sender, obtaining random messages r_0 and r_1.
   - The client sets its share of b as $\langle b \rangle_1^2 = r_0 \oplus r_1$.
   - The server sets its share of a as $\langle a \rangle_0^2 = d$.
   - The relationship $r_d = r_0 \oplus (\langle a \rangle_0^2 \wedge \langle b \rangle_1^2)$ holds.
2) **Second RC-ROT Instance**:
   - The client acts as the receiver, obtaining a random choice bit $e \in 0, 1$ and the corresponding random message s_e.
   - The server acts as the sender, obtaining random messages s_0 and s_1.
   - The server sets its share of b as $\langle b \rangle_0^2 = s_0 \oplus s_1$.
   - The client sets its share of a as $\langle a \rangle_1^2 = e$.
   - The relationship $s_e = s_0 \oplus (\langle a \rangle_1^2 \wedge \langle b \rangle_0^2)$ holds.
3) **Computing Shares of** c:

$$\langle c \rangle_0^2 = \langle a \rangle_0^2 \wedge \langle b \rangle_0^2 \oplus r_d \oplus s_0 \tag{4}$$

$$\langle c \rangle_1^2 = \langle a \rangle_1^2 \wedge \langle b \rangle_1^2 \oplus s_e \oplus r_0 \tag{5}$$

These computations ensure that when the parties combine their shares, they obtain $c = a \wedge b$, while keeping their individual inputs private.

Algorithm 4 presents the protocol for the secure AND functionality. This algorithm is a straightforward implementation of the GMW protocol [4] utilizing preprocessed Beaver's bit triples. Both the parties first retrieve a preprocessed bit triple to start the computation. They then compute local correction bits by XORing ($\oplus$) their input shares with the triple shares. These correction bits are exchanged and combined to reconstruct e and f. Finally, each party computes its output share $\langle z \rangle_p^2$ using the reconstructed correction bits, the triple shares, and the bit-complement of party index p', i.e. $p' = 1 - p$, ensuring that the underlying secret value in the shares is the logical AND of the inputs.

This protocol allows the parties to compute the AND of their secret-shared bits without revealing their private inputs, forming a building block for more complex secure computations.

D. 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 5.

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.

E. Average Pooling

Average pooling is an operation commonly used in neural networks to reduce the spatial dimensions of feature maps while retaining important information. In a secure inference setting, we need to perform average pooling on secret-shared data without revealing the underlying values. Algorithm 6 presents the protocol for average pooling adapted from CrypTFlow2 [38], which is employed in our system.

The protocol operates on secret shares of the input and produces secret shares of the output. The input matrix $\langle \mathbf{I} \rangle_p^{2^b}$ contains secret shares of the input values, where b is the bit width. The protocol initializes the output vector by assigning the first element from each window. It then iteratively accumulates the remaining elements in each window by performing secure addition on the secret shares. Finally, the protocol performs a division by the window size w using the DIV protocol from CrypTFlow2, which securely computes the division of secret-shared values by a public constant.

The division protocol DIV takes as input secret shares of the dividend and a public divisor and returns secret shares

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

Input: Output size n ; Flattened window size w
Input secret share array $\langle \mathbf{I} \rangle_p^{2^b}$ of size $n \times w$
Output: Flattened output secret share $\langle \mathbf{o} \rangle_p^{2^b}$

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

of the quotient. This operation is performed securely without revealing the intermediate sums or the final averaged values. By employing this protocol, we securely compute the average pooling operation on secret-shared data, which is essential for privacy-preserving neural network inference.

F. RLWE-HE and NTT

Homomorphic encryption (HE) enables computations on encrypted data without requiring decryption keys or access to the original plaintext. This approach allows a client to encrypt data, send it to a server for computation, and then decrypt the results. In HE systems, encryption and decryption are managed by the client, while the majority of the computational work is offloaded to the server. This division of labor reduces communication overhead, with data transfer involving only input and output sizes. Fully Homomorphic Encryption (FHE) schemes, introduced by Gentry [5], support arbitrary encrypted computations through a bootstrapping process but come with significant computational costs. Leveled HE schemes improve computational efficiency by allowing a limited number of operations without bootstrapping, although they increase ciphertext size. The capacity of these schemes is determined by their multiplicative depth, which indicates the maximum number of sequential multiplications.

The construction of most popular HE schemes, such as BGV [10], BFV [11], [12], and CKKS [13], relies on *Ring Learning With Errors* (RLWE) [25]. RLWE-based HE schemes operate on polynomial rings, denoted as $\mathcal{R}_Q^N$, where N represents the *polynomial modulus degree* and Q represents the *coefficient modulus*. In schemes such as BFV and BGV, plaintexts are elements of $\mathcal{R}_P^N$, with P ($< Q$) serving as the plaintext modulus. In CKKS, plaintexts are elements in $\mathcal{R}_Q^N$, without a plaintext modulus, facilitating approximate arithmetic.

Encryption in RLWE-based schemes involves encoding a secret vector $\mathbf{m} \in \mathbb{Z}_P^N$ into a polynomial $\overline{\mathbf{pt}}_\mathbf{m}$. The encryption yields a ciphertext $\overline{\mathbf{ct}}_\mathbf{m} = (\overline{\mathbf{a}}, \overline{\mathbf{b}})$, where $\overline{\mathbf{a}}$ is a random polynomial in the polynomial ring $\mathcal{R}_Q^N$, and $\overline{\mathbf{b}}$ is a polynomial in $\mathcal{R}_Q^N$ s.t.:

$$\overline{\mathbf{b}} = \left(\overline{\mathbf{a}} \cdot \overline{\mathbf{sk}} + \delta \cdot \overline{\mathbf{pt}}_\mathbf{m} + \epsilon \cdot \overline{\mathbf{e}}\right)$$

Here, $\overline{\mathbf{sk}} \in \mathcal{R}_{\{-1,0,1\}}^N$ represents the secret key, which is a polynomial with uniformly random ternary coefficients, $\delta, \epsilon \in \mathbb{Z}_Q$ are scheme-defined constants used to scale the plaintext and control noise tolerance, respectively, and $\overline{\mathbf{e}} \in \mathcal{R}_Q^N$ is a small error polynomial drawn from a zero-mean discrete Gaussian distribution with standard deviation σ. This small error $\overline{\mathbf{e}}$ is crucial for maintaining the hardness of the RLWE problem, thus ensuring the security of the encryption.

In HE, operations are performed on the ciphertexts by manipulating the underlying polynomials. Addition, multiplication, and rotation are the key operations supported by RLWE schemes. Addition and rotation lead to additive error growth, whereas multiplication results in multiplicative error growth, making parameter selection critical to control the error expansion.

HE multiplication corresponds to computing a convolution of the polynomial coefficients. Specifically, multiplying two polynomials $\overline{\mathbf{a}}$ and $\overline{\mathbf{b}}$ yields a result $\overline{\mathbf{c}} = \overline{\mathbf{a}} \times \overline{\mathbf{b}}$, where each coefficient c_i is determined through the convolution of the coefficients of $\overline{\mathbf{a}}$ and $\overline{\mathbf{b}}$:

$$c_i = \sum_{j=0}^{i} a_j \cdot b_{i-j} \pmod{Q}$$

This direct convolution has a quadratic complexity of $O(N^2)$, which can be computationally prohibitive for polynomials of high degree.

To mitigate this complexity, the *Number Theoretic Transform* (NTT) is used, which is analogous to the Fast Fourier Transform (FFT) but tailored for modular arithmetic, suitable for cryptographic applications. The NTT allows polynomials to be transformed into a point-value representation, where convolution transforms into pointwise multiplication. This reduces the complexity of polynomial multiplication from $O(N^2)$ to $O(N \log N)$. The NTT uses a "primitive root of unity" under modulo Q, unlike the FFT which employs complex roots of unity.

NTT-based multiplication proceeds in three main steps. First, both polynomials $\overline{\mathbf{a}}$ and $\overline{\mathbf{b}}$ are transformed using the NTT, resulting in $\mathrm{NTT}(\overline{\mathbf{a}})$ and $\mathrm{NTT}(\overline{\mathbf{b}})$. The transformation is applied to convert the polynomial from its coefficient representation to a point-value representation in $\mathcal{R}_Q^N$. Second, pointwise multiplication is performed – one scalar multiplication per coefficient:

$$\mathrm{NTT}(\overline{\mathbf{c}})_i = \mathrm{NTT}(\overline{\mathbf{a}})_i \cdot \mathrm{NTT}(\overline{\mathbf{b}})_i \pmod{Q}$$

Finally, an inverse NTT is applied to obtain the product $\overline{\mathbf{c}}$ back in the coefficient domain, $\overline{\mathbf{c}} = \mathrm{NTT}^{-1}(\mathrm{NTT}(\overline{\mathbf{c}}))$.

The use of the NTT not only reduces computational costs but also ensures that operations remain consistent with the modular arithmetic required for cryptographic security. The advantage of the NTT in HE is that it optimizes the core operation of polynomial multiplication, transforming the otherwise convolution-heavy multiplication into a sequence of

efficient element-wise multiplications. Given that polynomial multiplication is central to encrypted computation, this optimization is essential for making homomorphic encryption schemes practical and efficient. The use of modular arithmetic throughout ensures that all operations are secure and stay within the defined finite field, which is critical for maintaining the integrity and security of HE schemes.

G. ResNet50 Evaluations

In Figure 7 we show the results of our evaluations on the ResNet50 model with 32 Threads. SecONNds significantly outperforms existing frameworks in both runtime and communication efficiency. Specifically, SecONNds achieves a total runtime of 25 seconds, demonstrating a $1.9\times$ speedup over Cheetah's 49 seconds and a substantial $7\times$ improvement compared to CrypTFlow2's 180 seconds. In terms of communication volume, SecONNds reduces data transfer to 2061 MiB, marking an 8.5% decrease from Cheetah's 2248 MiB.

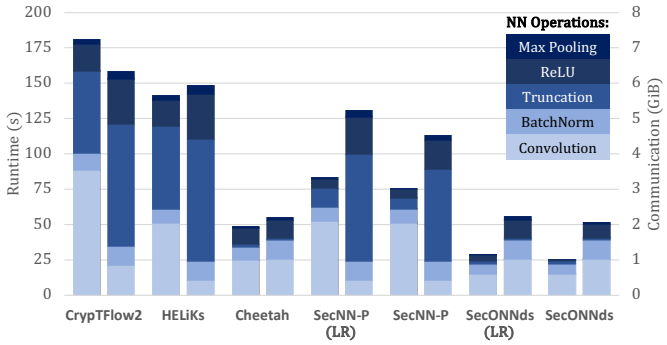

Fig. 7. End-to-end (E2E) runtime (left bar) and communication (right bar) performance of each framework for Neural Network operations in ResNet50.

The SecONNds-P variants maintain competitive performance, balancing runtime and communication efficiency across different configurations. These results underscore SecONNds' effectiveness in optimizing secure inference for complex neural network architectures like ResNet50.