TABULA: EFFICIENTLY COMPUTING NONLINEAR ACTIVATION FUNCTIONS FOR PRIVATE NEURAL NETWORK INFERENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Multiparty computation approaches to private neural network inference require significant communication between server and client, incur tremendous runtime penalties, and cost massive storage overheads. The primary source of these expenses is garbled circuits operations for nonlinear activation functions (typically ReLU), which require on the order of kilobytes of data transfer for each individual operation and tens of kilobytes of preprocessing storage per operation per inference. We propose a replacement for garbled circuits: TABULA, an algorithm to securely and efficiently perform single operand nonlinear functions for private neural network inference. TABULA performs a one time client initialization procedure with the help of a trusted third party (or via using fully homomorphic encryption), operates over smaller finite fields whose elements are representable with less than 16 bits, and employs a lookup table which stores the encrypted results of nonlinear operations over secretly shared values. We show TABULA is secure under a semi-honest threat model, allowing it to be used as a replacement for garbled circuits operations. Our results show that for private neural network inference, TABULA eliminates communication by a factor of more than $50\times$, enables speedups over $10\times$, and reduces storage costs from $O(n)$ to $O(1)$.

1 INTRODUCTION

Private neural network inference seeks to allow a server to perform neural network inference on a client's inputs while minimizing the data leakage between the two parties. Concretely, the server holds a neural network model M while the client holds an input x and the objective of a private inference protocol is for the client to obtain $M(x)$ without revealing any information about the client's input x to the server, nor revealing any information about the server's model M to the client. A protocol for private neural network inference brings tremendous value to both the server and the clients: the clients' sensitive input data is kept secret from the server and shields the user from malicious data collection, and additionally, the server's model is protected from the client and prevents it from being reverse engineered or stolen by competitors.

Current state-of-the-art multiparty computation approaches to private neural network inference require significant communication between client and server, lead to massive runtime slowdowns, and incur tremendous storage penalties (Mishra et al., 2020a; Ghodsi et al., 2021; Jha et al., 2021; Rathee et al., 2020; Juvekar et al., 2018; Cho et al., 2021). The source of these expenses is computing nonlinear activation functions with garbled circuits, a generic multiparty computation method for securely computing functions over secret inputs (Yao, 1986). Garbled circuits not only require significant storage for preprocessed circuits, but also require multiple rounds of communication at inference time to encrypt, decrypt, and execute the nonlinear function. Concretely, individual ReLUs implemented as garbled circuit operations require over 2 KB of communication per scalar element during inference (Mishra et al., 2020a) and cost over 17 KB of preprocessing storage per scalar element for each individual inference (Mishra et al., 2020a; Ghodsi et al., 2021). These costs make state-of-the-art neural network models prohibitively expensive to deploy: on ResNet-32, state-of-the-art multiparty computation approaches for a single private inference require greater than 60 MB of data communication (Mishra et al., 2020a), take more than 10 seconds for an individual inference (Mishra et al., 2020a), and cost over 5 GB of preprocessing storage per inference (Ghodsi et al.,

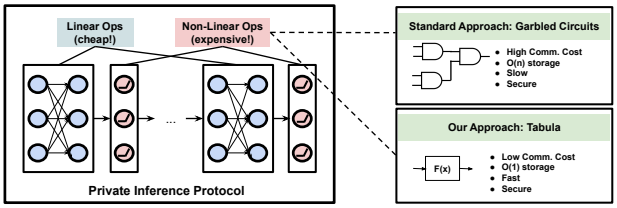


Figure 1: TABULA replaces garbled circuits in standard private neural network inference protocols. Compared to garbled circuits, TABULA obtains lower communication costs, better storage savings, and higher speedups while maintaining security. Source code is released at https://github.com/tabulainference/tabula.

2021). These communication, runtime and storage costs pose a significant barrier to deployment, as they degrade user experience, drain clients' batteries, induce high network expenses, and eliminate applications that require sustained real time inference.

We introduce TABULA, an algorithm to efficiently and securely perform nonlinear activation functions for private neural network inference, eliminating garbled circuits from their computational mix (see Figure 1). TABULA performs a one time client initialization procedure with the help of a trusted third party (or by using fully homomorphic encryption) to precompute a large lookup table containing the encrypted result of nonlinear operations over secret shares. This initialization procedure is done exactly once per client, who generates a private secret key that subsequently allows them to securely perform numerous queries to this table. During online inference, computing a nonlinear function is just a single lookup to the precomputed table, requiring only one round of communication. TABULA's table lookup approach to securely computing nonlinear functions is uniquely suited to computing neural network activation functions as neural networks may be quantized with little loss in accuracy, and additionally activation functions are single operand; these properties enable employing a table lookup approach to securely and efficiently compute neural network nonlinear activation functions without requiring an infeasibly large table size. Our contributions are as follows:

- We develop TABULA, a secure and efficient algorithm for performing nonlinear functions for private neural network inference. For ReLU, TABULA reduces inference-time communication from 16,000 bits to 150 bits per scalar element in exchange for a one time client initialization procedure with involvement from a trusted third party and the use of an 4-32 GB lookup table located on the server. Experiments on LeNet for MNIST, ResNet-32 for Cifar-10 and ResNet-34 for Cifar-100 show that TABULA reduces communication by over $100\times$, runtime by more than $10\times$, and storage by greater than $10\times$.
- We show that TABULA provides security for parties' data to the extent that an 128-bit secret key is guessable by an adversary. Specifically, we prove that TABULA inputs and outputs are uniformly and randomly distributed in $\mathbb{F}_p$ and hence that under a semi-honest threat model TABULA leaks no information to either party. This allows TABULA to be used as a replacement for garbled circuits for single-operand nonlinear operations where the finite field is small enough (i.e: when values of the finite field are representable by < 16 bits).

2 RELATED WORK

2.1 MULTIPARTY COMPUTATION APPROACHES TO PRIVATE NEURAL NETWORK INFERENCE

Multiparty computation approaches to private neural network inference have traditionally been limited by both computation and communication, and prior lines of work focus on reducing these systems costs while maintaining privacy during execution. Early secure machine learning systems like SecureML (Mohassel & Zhang, 2017; Rouhani et al., 2017; Rathee et al., 2020; Keller, 2020) address the issue of secure computation for simpler, linear machine learning models like logistic regression and use traditional multiparty computation techniques (Keller, 2020) in their algorithms. More recently, specialized systems have emerged that specifically target private neural network inference, including works like Minionn, Gazelle and Delphi (Liu et al., 2017; Juvekar et al., 2018; Mishra et al., 2020a; Lehmkuhl et al., 2021; Rathee et al., 2020). These works have successively

optimized the linear portions of private neural network inference (via techniques like preprocessing linear layers with homomorphic encryption) to the point where linear operations for private neural network inference are effectively free in terms of runtime during inference (Mishra et al., 2020a). Recent works to reduce the cost of neural network nonlinear activation functions include DeepReduce, Delphi, Sphynx and Circa (Jha et al., 2021; Mishra et al., 2020a; Cho et al., 2021; Ghodsi et al., 2021), which focus on learning neural network architectures that minimize the use of nonlinear activation functions and designing more efficient circuits for ReLU. However, while effective, these techniques invariably run into the fundamental costs imposed by garbled circuits.

Unlike prior approaches to mitigate the cost of nonlinear activation functions for private neural network inference, our approach directly addresses the problems posed by garbled circuits by eliminating them altogether. Our method is centered around precomputing a large lookup table containing the encrypted results of nonlinear activation functions on secret shares, and using quantization to reduce the size of this table. Our approach can be applied with prior approaches that are focused on designing neural architectures that minimize the use of nonlinear activation functions, like Delphi's planner and DeepReduce.

2.2 Lookup Tables for Secure Computation

Lookup tables have been used to speed up computation for applications in both secure multiparty computation (Launchbury et al., 2012; Damgård et al., 2017; Keller et al., 2017; Rass et al., 2015; Dessouky et al., 2017) and homomorphic encryption (Li et al., 2019; Crawford et al., 2018). These works have demonstrated that lookup tables may be used as an efficient alternative to garbled circuits, provided that the input space is small. Prior works have primarily focused on using lookup tables to speed up traditional applications like computing AES (Keller et al., 2017; Damgård et al., 2017; Launchbury et al., 2012; Dessouky et al., 2017) and data aggregation (Rass et al., 2015) (with the exception of Crawford et al. (2018) which focuses on linear regression).

To the best of our knowledge, little prior work has been done to use lookup tables with PRFs to securely and efficiently compute nonlinear activation functions in the execution of relatively large neural networks, and current state of the art private inference systems like Delphi (Mishra et al., 2020b), SecureML (Mohassel & Zhang, 2017), Circa (Ghodsi et al., 2021), DeepReduce Jha et al. (2021) all still use garbled circuits. Notably, a significant limiting factor to the lookup table approach for secure computation is its exponential space requirements (which grows exponentially with the number of operands and their precisions), which we suspect limits its use in other applications. The lookup table approach is uniquely well suited to securely and efficiently compute neural network nonlinear activation functions for two reasons: 1) neural network inference can operate over low precision numbers and tolerates noise with little degradation to accuracy and 2) neural network activation functions are single operand. These two factors allow us to limit the size of the lookup table to be sufficiently small to be practical, and consequently we can achieve the significant performance benefits of lookup tables at runtime (i.e order of magnitude less communication and storage).

3 Tabula: Efficient Nonlinear Activation Functions for Private Neural Network Inference

Tabula is a secure algorithm for efficiently computing single operand nonlinear activation functions for private neural network inference and operates over secret shares of the input. Tabula is designed to work as a replacement for garbled circuits within the canonical framework of multiparty computation methods for private inference. In the background, we state the goals of private inference, define the basic cryptographic primitives used for Tabula and outline how standard multiparty computation protocols operate. Then we detail the algorithmic workings of Tabula, explain how it fits into standard multiparty computation protocols, prove the security of the protocol, and analyze its communication and storage requirements.

3.1 Background

Private Inference Objectives, Threat Model
Private neural network inference seeks to compute a sequence of linear and nonlinear operations parameterized by the server's model over a client's input while revealing as little information to

either party beyond the model's final prediction. Formally, given the server's weights W_i and the client's private input x, the goal of private neural network inference is to compute

$$a_i = A(W_i a_{i-1})$$

where $a_0 = x$ and A is a nonlinear activaiton function, typically ReLU.

State-of-the-art private neural network inference protocols like Delphi operate under a two-party semi-honest setting (Mishra et al., 2020a; Lehmkuhl et al., 2021) (with notable exceptions such as Muse (Lehmkuhl et al., 2021)), where only one of the parties is corrupted and the corrupted party follows protocol. Importantly, these private inference protocols do not protect the architecture of the neural network being executed (only its weights), and do not secure any information leaked by the predictions themselves (Mishra et al., 2020a). As we follow Delphi's protocol (described two sections down) for the overall private execution of the neural network (TABULA only handles the nonlinear portions of the protocol), these security assumptions are implicitly assumed.

Cryptographic Primitives, Notations, Definitions
TABULA utilizes standard tools in multiparty computation. Multiparty computation methods operate over finite fields and we define $\mathbb{F}_p$ as a finite field over prime p with n bits. Additionally, we use $[x]$ to denote a two party secret sharing of the scalar $x \in \mathbb{F}_p$: $x = [x]_0 + [x]_1$ where $[x]_i$ are independently and randomly distributed in $\mathbb{F}_p$. Additionally, like in prior works, we assume that negative values are encoded by numbers above $\frac{n-1}{2}$. Furthermore, TABULA utilizes basic constructs such as hash/lookup tables to store encrypted function results. We define H as a hash lookup table that supports basic insertions and queries. More specifically, $H[i] = j$ sets the table's key i to the value j, and future queries to the key i would return j. Finally, TABULA leverages basic tools in cryptography. Specifically, TABULA heavily utilizes PRFs (pseudo random function families). We define $F_k(x)$ as a PRF over key k and input x. The outputs of $F_k(x)$ over a sequence of inputs is indistinguishable from the outputs of a truly random function regardless of how the inputs were chosen and given that k is random. More formally, $\{F_k : \mathbb{F}_p \rightarrow \mathbb{F}_p\}_{k \in \{0,1\}^z, z>n}$ such that F_k is computable in polynomial $poly(n)$ time and for any adversary $\mathcal{A}$ we have

$$Adv(\mathcal{A}) = |Pr[\mathcal{A}^{F_k(\cdot)} = 1] - Pr[\mathcal{A}^{R(\cdot)} = 1]| \leq negl(n)$$

Delphi Private Inference Protocol
To understand how TABULA fits into standard private neural network inference protocols like Delphi (Mishra et al., 2020b), we briefly outline how these protocols operate. Broadly, state-of-the-art private inference protocols are divided into a per-input preprocessing phase and an online inference phase. In our work, we build on top of the Delphi private inference framework (Mishra et al., 2020a), which operates as following.

- **Per-Input Preprocessing Phase**
  This phase preprocesses data to prepare for the secure execution of a single input, and is performed to make the online inference phase faster. For each linear linear layer, this process entails the client generating and encrypting a random vector $r_c \in \mathbb{F}_p \rightarrow Enc_k(r_c)$ with linearly homomorphic encryption (k is the public key), sending it to the server to compute $Enc_k(W_i r_c + r_s)$ where $r_w \in \mathbb{F}_p$ is also randomly generated for that particular layer, which is sent back to the client who obtains $W_i r_c + r_s$. This procedure enables the use of standard linear operations at inference time instead of homomorphic encryption operations. For nonlinear layers, the server garbles the labels of the ReLU circuits and sends them to the client.
- **Online Inference Phase**
  This phase performs the actual inference on a client's input. For linear layers, the client and server start with $[x]_0$, $[x]_1$ respectively. The client adds $[x]_0$ with that layer's r_c to obtain $[x]_0 + r_c$, sends it to the server so that it obtains $x + r_c$, which then computes $W_i(x + r_c) + r_s = W_i x + W_i r_c + r_s$ (the r_s for that particular layer). At this point, client and server hold the secret sharing $[W_i x]$. Hence, the input to the nonlinear activation function is a secret share $[a] = [W_i x]$. For nonlinear layers, particularly ReLU, the client and server perform the garbled circuits operation, which outputs the secret share of the inputs for the next layer.

Hence, to replace the nonlinear portions of this protocol, we need to construct a function that takes in $[x]$ and securely computes and outputs $[ReLU(x)]$.

3.2 TABULA FOR NEURAL NETWORK NONLINEAR ACTIVATION FUNCTIONS

We present our algorithm for secure and efficient computation of nonlinear activation functions below. Our algorithm is divided into a one time client initialization procedure with involvement from a trusted third party, and the online nonlinear execution phase.

Problem Statement
We aim to construct a function $T : \mathbb{F}_p \rightarrow \mathbb{F}_p$ which given $[x]$ securely computes $[A(x)] : T([x]) = [A(x)]$ where $A : \mathbb{F}_p \rightarrow \mathbb{F}_p$ is a nonlinear single operand function. Concretely, security implies that all communication (across multiple function calls, regardless of the values of x) between client and server in this procedure is indistinguishable from uniform random noise in $\mathbb{F}_p$.

Client Initialization
TABULA performs a one time client initialization phase to precompute a lookup table containing the encrypted results of the nonlinearity over every possible secret sharing of $[x]$. This process requires involvement from a trusted third party (to both client and server) to compute. The steps of the initialization phase is as follows:

Algorithm 1 TABULA Client Initialization

1: Client and server generate a 128 bit secret $s_c \in \{0,1\}^{128}$, $s_s \in \{0,1\}^{128}$ respectively
2: Client generates another 128 bit secret $d_c \in \{0,1\}^{128}$
3: Trusted third party initializes H
4: **for** $x \in \mathbb{F}_p$, $y \in \mathbb{F}_p$ **do**
5: Trusted third party obtains $a = F_{s_c}(x)$ from client, $b = F_{s_s}(y)$ from server
6: Trusted third party sets $H[a||b] = (A(x+y) + F_{d_c}(x)) \bmod p$
7: **end for**
8: Trusted third party sends H to server

We note that while we state that this procedure requires involvement from a trusted third party, this procedure can also be performed with fully homomorphic encryption (SEAL). The change for this is simply to iterate over x_i and y_i pairs in a random permutation, encrypt and perform all operations with fully homomorphic encryption, index by $F_k(a||b)$ instead of just $a||b$, then decrypt the key and value before inserting into the table. This process can be extremely compute intensive and we highlight that this is a one time cost per client (rather than per-input as in preprocessing phases for standard private inference protocols). Other potential approaches might rely on more advanced techniques such as verifiable computation (Goldwasser et al., 2015; Thaler et al., 2012). We leave developing more efficient initialization procedures for this component (with or without the help of a trusted third party) to future research.

Online Execution
Given the preprocessing step has been performed, the online execution step for TABULA, T, is straightforward and outlined in Algorithm 2.

Algorithm 2 TABULA Online Execution

Input: Client holds $[x]_0$, Server hold $[x]_1$
Output: Client holds $[A(x)]_0$, Server holds $[A(x)]_1$

1: Client computes $F_{s_c}([x]_0)$ and sends to server
2: Server computes $F_{s_s}([x]_1)$
3: Server looks up $a = H[F_{s_c}([x]_0)||F_{s_s}([x]_1)]$
4: Server generates random $r \in \mathbb{F}_p$
5: Server returns to client $a + r = (A(x) + F_{d_c}([x]_0) + r) \bmod p$
6: Client obtains $A(x) + r$ by subtracting $F_{d_c}([x]_0)$

At the end, client and server hold r and $A(x) + r$ and hence hold a secret sharing of $[A(x)]$. Note that, across multiple function calls, the same secrets s_c, s_s, d_c and H are used.

Security
We show that our method is secure under the semi-honest setting, specifically, that all commu-

nication between server and client during the online execution phase is uniformly and randomly distributed in $\mathbb{F}_p$ across multiple function calls and hence no information is learned by the client or server in this process. Concretely, to prove security, we need to ensure that 1) the client's encrypted index $F_{s_c}([x]_0)$ sent to the server is uniformly randomly distributed in $\mathbb{F}_p$, 2) the result of the lookup $a = H[F_{s_c}([x]_0)||F_{s_s}([x]_1)]$ that the server obtains is uniformly randomly distributed in $\mathbb{F}_p$ and 3) the data the server returns to the client $a+r$ is uniformly randomly distributed in $\mathbb{F}_p$. Note that, as stated in the background, we presume a semi-honest setting: this is important as it ensures that across multiple private inferences $[x]_i$ is uniformly distributed across $\mathbb{F}_p$; without this property, the distribution of communication from client to server is no longer uniform and hence would leak information.

First, we show that communication sent by server to client is secure.

Theorem 1. *Across one or multiple invocations of $T : \mathbb{F}_p \rightarrow \mathbb{F}_p$, all communication from server to client is uniformly randomly distributed in $\mathbb{F}_p$.*

Proof. This follows from the fact that all values sent from server to client are additively blinded by r which is randomly distributed in $\mathbb{F}_p$ upon every invocation of T. Hence, all data sent from server to client in the online phase across multiple calls of T are indistinguishable from random noise and leaks no information. □

Next, we show that the key lookup procedure from client to server is secure.

Theorem 2. *Across one or multiple invocations of $T : \mathbb{F}_p \rightarrow \mathbb{F}_p$, $F_{s_c}([x]_0)$ is uniformly and randomly distributed in $\mathbb{F}_p$.*

Proof. This follows from the randomness property of $F_{s_c}([x]_0)$ and that $[x]_0$ is uniformly distributed in $\mathbb{F}_p$ across multiple invocations of T. □

Finally, we show that no information is leaked to the server by the result of the lookup $A(x) + F_{d_c}([x]_0)$ mod p. To do this, we first show that $A(x) + F_{d_c}([x]_0)$ mod p is a PRF.

Theorem 3. *Suppose $\{F_k : \mathbb{F}_p \rightarrow \mathbb{F}_p\}_{k \in \{0,1\}^z, z>n}$ is a secure PRF such that for any adversary $\mathcal{A}$, $|Pr[\mathcal{A}^{F_k(\cdot)} = 1] - Pr[\mathcal{A}^{R(\cdot)} = 1]| \leq negl(n)$. Then $F_k^1(x, r) = A(x) + F_k(x + r)$ mod p is a secure PRF such that for every adversary $\mathcal{A}$, $|Pr[\mathcal{A}^{F_k^1(\cdot)} = 1] - Pr[\mathcal{A}^{R(\cdot)} = 1]| \leq negl(n)$.*

Proof. We prove this via contradiction by showing that if the statement were not true, then F_k is not a secure PRF. Suppose that there exists an adversary $\mathcal{A}_{F^1}$ such that $|Pr[\mathcal{A}_{F^1}^{F_k^1(\cdot)} = 1] - Pr[\mathcal{A}_{F^1}^{R(\cdot)} = 1]| > negl(n)$. Then define adversary $\mathcal{A}_F$ to have the following routine

Algorithm 3 ADVERSARY $\mathcal{A}_F$

1: Start running adversary $\mathcal{A}_{F^1}$
2: Whenever $\mathcal{A}_{F^1}$ makes a query (x,r), forward it to the oracle $y = F'(x + r)$ (who either always returns $F_k(x + r)$ or $R(:)$). Give back to $\mathcal{A}_{F^1}$, $A(x) + y$ mod p
3: Output whatever $\mathcal{A}_{F^1}$ outputs

We see that $Pr[\mathcal{A}_F^{R(\cdot)} = 1] = Pr[\mathcal{A}_{F^1}^{R(\cdot)} = 1]$ as $A(x) + R(:)$ mod p is randomly distributed in $\mathbb{F}_p$. Furthermore $Pr[\mathcal{A}_F^{F_k(\cdot)} = 1] = Pr[\mathcal{A}_{F^1}^{F_k^1(\cdot)} = 1]$ as when $F' = F_k$ we pass to $\mathcal{A}_{F^1}$, $F_k^1(x, r) = A(x) + F_k(x + r)$ mod p. Hence $Adv(\mathcal{A}_F) = Adv(\mathcal{A}_{F_1}) > negl(n)$. □

Hence, the output of H is a PRF and as the input to the lookup table is uniformly distributed (the concatenation of two PRF outputs is also a PRF, with uniform inputs), then so too is the output. Thus, we have shown that all inputs to T sent from client to server are uniformly randomly distributed, all outputs of the lookup table of T are uniformly randomly distributed, and all data sent back to the client by the server are uniformly randomly distributed, ensuring that no information is leaked to either party during the execution of T, with the assumption that the secrets s_c, s_s, d_c are unknown to the opposite party.

TABULA Communication and Storage Costs
TABULA incurs a one round communication cost between client and server to look up the result of the nonlinearity and to obtain the secret share of the result. The communication cost incurred by the key lookup is ≤ 128 bits per nonlinearity as the output of P_k is ≤ 128 bits per nonlinearity. The communication cost incurred by returning the secret share is n bits (the number of bits of prime p). Thus, the total communication cost of TABULA is $\leq 128 + n$. Note that in our implementation, we truncate the output of P_k to be $2n$ bits to minimize communication and also to eliminate collisions (note that truncation of a PRF is still a PRF); with $n = 16$, the total communication cost is 48 bits or 6 bytes per ReLU.

TABULA storage costs depend on the number of elements in H and the size of each element. As H contains every possible 2-combination of $[x]$ and each value is n bits, we see that H will be of size $2^{2n} \times (n + size(k))$ bits where $size(k)$ is the size in bits of the key. As indicated, H becomes exponentially larger with increasingly large fields, hence the size of the finite field must be small enough to ensure H is reasonably sized. Fortunately, various recent works in neural network quantization have shown that neural networks can operate with < 16 bit precision for operations (Ni et al., 2020; de Bruin et al., 2020; Zhao et al., 2020; McKinstry et al., 2019), enabling us to use finite fields where $n < 16$ bits. Hence, with $n < 16$ the size of H is less than or equal to 18 GB (assuming key size of 128 bits), which reasonably fits on workstation class machines.

4 RESULTS

We present the benefits of TABULA over state-of-the-art protocols that use garbled circuits for nonlinear activation functions, specifically ReLU. We evaluate our method on neural networks including a large variant of LeNet for MNIST, ResNet-32 for Cifar10, and ResNet-34 / VGG-16 for Cifar-100, which are relatively large image recognition neural networks that prior private inference works benchmark (Ghodsi et al., 2021; Mishra et al., 2020a; Jha et al., 2021; van der Hagen & Lucia, 2021). Unless otherwise stated, we compare against our implementation of the Delphi protocol (Mishra et al., 2020a) using garbled circuits for nonlinear activation functions, without neural architecture changes, during the online inference phase. The Delphi protocol with garbled circuits, to the best our knowledge, is the current state of the art for private neural network inference. Experiments are run on AWS c5.4x large machines (US-West1 (N. California) and US-West2 (Oregon)) which have 8 physical Intel Xeon Platinum @ 3 GHz CPUs and 32 GiB RAM; network bandwidth between these two machines achieves a maximum of 5-10 Gbit/sec, according to AWS. Note that we use the same machine/region specs as detailed in Mishra et al. (2020a), but with 2x more cores/memory (c5.4xlarge vs c5.2xlarge). We note that further results are included in the appendix.

4.1 COMMUNICATION REDUCTION

ReLU Communication Reduction
We benchmark the amount of communication required to perform a single ReLU with garbled circuits vs TABULA. Table 1 shows the amount of communication required by both protocols during online inference. TABULA achieves more than a $100\times$ reduction in communication over garbled circuits. Our implementation of garbled circuits obtains the same same communication cost as reported by Mishra et al. (2020a).

Garbled Circuits	TABULA	Communication Reduction
2KB	6B	$> 340\times$

Table 1: TABULA vs garbled circuits communication cost for a single ReLU operation.

End-to-end Communication Reduction on LeNet, Resnet-32, ResNet-34
We benchmark the total amount of communication required during the online phase of a single private inference for various network architectures (batch size 1). Table 1 shows the number of ReLUs per network, as well as the communication costs of using garbled circuits vs TABULA. TABULA obtains $> 50\times$ reduction in communication across various network architectures when compared to this lower bound. Note that these numbers reflect total communication costs (not just

ReLU communication costs); further note that we do not make any architectural changes to the neural network (e.g: replace any ReLU operations with quadratic operations, etc).

Network	# ReLUs	Garbled Circuits	TABULA	Communication Reduction
LeNet	58K	60 MB	931 KB	$> 65\times$
VGG-16	276K	286 MB	4.2 MB	$> 68\times$
ResNet-32	303K	311 MB	4.7 MB	$> 66\times$
ResNet-34	1.47M	1.5 GB	21 MB	$> 73\times$

Table 2: TABULA vs garbled circuits total communication cost during private inference for different network architectures.

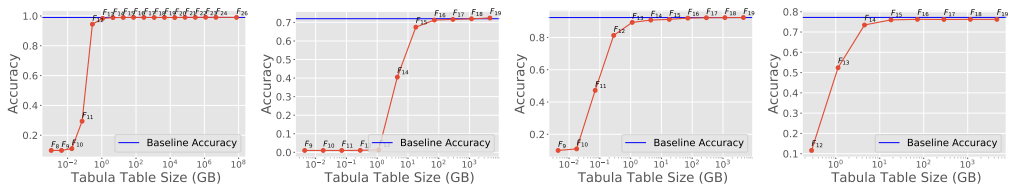
4.2 STORAGE SAVINGS

ReLU Storage Savings
We compare the amount of extra storage required to prepare for a single ReLU with garbled circuits vs TABULA. Table 3 shows the amount of preprocessing storage required by garbled circuits vs TABULA, with different circuit implementations for garbled circuits (Delphi's (Mishra et al., 2020a) and Circa's (Ghodsi et al., 2021)). As shown, garbled circuits requires significant extra preprocessing storage for each ReLU, that scales linearly with the number of inferences to be performed, whereas TABULA requires none. Instead, TABULA requires the storage of a single lookup table containing the results of the nonlinear activation function; we present these storage costs in the next section.

Garbled Circuits (Delphi)	Garbled Circuits (Circa)	TABULA
17.5 KB	3.75 KB	0

Table 3: TABULA vs garbled circuits extra preprocessing storage cost per ReLU per inference. Garbled circuit storage costs obtained from Mishra et al. (2020a) and Ghodsi et al. (2021).

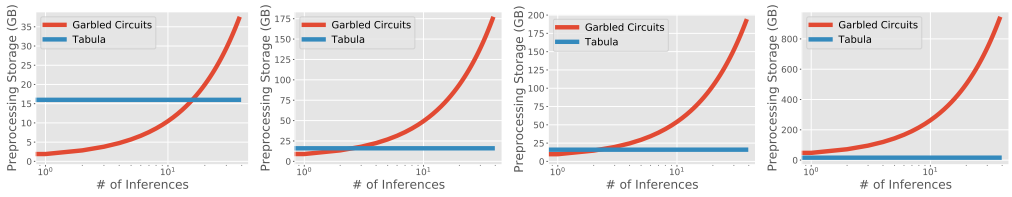
TABULA Table Size vs Accuracy
We benchmark the accuracy achieved for various networks/tasks versus how large TABULA's lookup table is allowed to grow, via reducing the size of the finite field of the network's weights via quantization. Recall TABULA'S lookup table size is a direct function of how many bits are used for scalars of the finite field during execution (e.g: if operating over $\mathbb{F}_p$ where p is 16 bits, then the lookup table size will be $2^{32} \times (2 + 16)$ bytes). Hence, we perform a random search to assign different precisions of finite fields of each layer of each network, then measure the maximum of the number of bits required to represent all weight/activation values seen during inference, comparing it with the accuracy achieved on the corresponding task (please see the supplemental for more details). Figure 2 shows that across all tasks, using a 15 bit finite field prime is sufficient to obtain within 1-4% of baseline accuracy, leading to a 18 GB lookup table. Garbled circuits which may use a 32 bit finite field prime would achieve at most full precision accuracy, and hence the 1-4% accuracy loss is an extra cost for TABULA. Note that, bigger networks may be more heavily quantized without loss of accuracy Zhou et al. (2019).



(a) LeNet MNIST (b) VGG-16 Cifar100 (c) ResNet-32 Cifar10 (d) ResNet-34 Cifar100

Figure 2: TABULA table size vs achieved accuracy on corresponding tasks. In this plot, we use F_b to represent the number of bits b of p of the finite field. Using a 15-bit finite field prime achieves within 4 % error of the baseline accuracy, leading to a table size of 18 GB.

End-to-end Storage Savings on LeNet, ResNet-32, ResNet-34
We compare the total amount of storage required for different numbers of inferences on different networks/tasks between TABULA and garbled circuits. Figure 3 shows that within 100 inferences, the storage cost of TABULA's 18 GB lookup table is greatly exceeded by the amount of storage required for garbled circuits. Note again that for every inference, new garbled circuits are generated to ensure security. Hence, garbled circuits require $O(n)$ storage with increasing number of inferences, whereas TABULA requires $O(1)$ storage. In Figure 3 we assume TABULA uses a 18 GB lookup table (15 bit prime for finite field), and garbled circuits cost 17.5 KB per relu per inference as reported in Delphi (Mishra et al., 2020a) and (Ghodsi et al., 2021).



(a) LeNet (b) VGG-16 (c) ResNet-32 (d) ResNet-34

Figure 3: End-to-end storage costs for TABULA vs garbled circuits, across multiple inferences.

4.3 END-TO-END RUNTIME SPEEDUP

We compare the total runtime speedups TABULA obtains over garbled circuits across various networks. Table 4 shows that TABULA obtains over $10\times$ speedup across different neural networks. Note that our implementation of garbled circuits obtains similar throughput as reported in Delphi (Mishra et al., 2020a): our implementation of garbled circuits takes 126 us per ReLU, whereas the reported is 84 us (Mishra et al., 2020a). As shown, bigger networks are increasingly bottlenecked by ReLU operations (as they have proportionally more of them), and hence TABULA's runtime reduction scales with the size of the neural network. For smaller networks with fewer ReLUs like LeNet, TABULA's runtime is no longer bottlenecked by communication but rather overheads like computing the PRF and table lookup; despite this, we still see significant speedups. Systems efforts to optimize the PRF and table lookups can be improved with software optimizations (like parallelizing the operation, writing it in C/assembly, leveraging SIMD, etc) or improved hardware (e.g: faster RAM, larger cache).

Network	**# ReLUs**	**Garbled Circuits Runtime (s)**	**TABULA Runtime (s)**	**Speedup**
LeNet	58K	7.8	.23	$> 33.9\times$
VGG-16	276K	35.7	1.0	$> 35.7\times$
ResNet-32	303K	48.5	1.8	$> 27.1\times$
ResNet-34	1.47M	181.8	4.3	$> 42.3\times$

Table 4: TABULA end-to-end runtime speedup compared with garbled circuits.

5 CONCLUSION

We propose TABULA, a secure and efficient algorithm for computing nonlinear activation functions for private neural network inference. Compared to garbled circuits, our results show that our method obtains significant reductions in communication, storage and runtime. With TABULA, networked communication costs are no longer the bottleneck for private neural network inference, and standard compute optimization techniques can be used to further reduce arithmetic and memory costs to obtain secure and efficient private neural network inference protocols. TABULA is a step towards sustained, low latency, low energy, low bandwidth real time private inference applications.

6 ETHICS STATEMENT

TABULA is a method for improving the efficiency of private neural network inference. Our algorithm contributes to the increased adoption of privacy preserving machine learning techniques, which we believe will have a positive impact on society by improving user data privacy.

7 REPRODUCIBILITY STATEMENT

All code and experiments for TABULA is or will be open sourced through Github. Additionally, implementation details are described in the appendix.

REFERENCES

Minsu Cho, Zahra Ghodsi, Brandon Reagen, Siddharth Garg, and Chinmay Hegde. Sphynx: Relu-efficient network design for private inference, 2021.

Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. Doing real work with fhe: The case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing; Applied Homomorphic Cryptography*, WAHC '18, pp. 1–12, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450359870. doi: 10.1145/3267973.3267974. URL https://doi.org/10.1145/3267973.3267974.

Ivan Damgård, Jesper Buus Nielsen, Michael Nielsen, and Samuel Ranellucci. The tinytable protocol for 2-party secure computation, or: Gate-scrambling revisited. In Jonathan Katz and Hovav Shacham (eds.), *Advances in Cryptology – CRYPTO 2017*, pp. 167–187, Cham, 2017. Springer International Publishing.

Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. Quantization of deep neural networks for accumulator-constrained processors. *Microprocessors and Microsystems*, 72:102872, 2020. ISSN 0141-9331. doi: https://doi.org/10.1016/j.micpro.2019.102872. URL https://www.sciencedirect.com/science/article/pii/S0141933119301164.

G. Dessouky, F. Koushanfar, A. Sadeghi, T. Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. *IACR Cryptol. ePrint Arch.*, 2018:486, 2017.

Zahra Ghodsi, Nandan Kumar Jha, Brandon Reagen, and Siddharth Garg. Circa: Stochastic relus for private deep learning, 2021.

Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. *J. ACM*, 62(4), September 2015. ISSN 0004-5411. doi: 10.1145/2699436. URL https://doi.org/10.1145/2699436.

Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. Deepreduce: Relu reduction for fast private inference, 2021.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. Gazelle: A low latency framework for secure neural network inference, 2018.

Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020. doi: 10.1145/3372297.3417872. URL https://doi.org/10.1145/3372297.3417872.

Marcel Keller, Emmanuela Orsini, Dragos Rotaru, Peter Scholl, Eduardo Soria-Vazquez, and Srinivas Vivek. Faster secure multi-party computation of aes and des using lookup tables. In *International Conference on Applied Cryptography and Network Security*, pp. 229–249. Springer, 2017.

John Launchbury, Iavor S. Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. ICFP '12, pp. 189–200, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450310543. doi: 10.1145/2364527.2364556. URL https://doi.org/10.1145/2364527.2364556.

Ryan Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca Ada Popa. Muse: Secure inference resilient to malicious clients. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2201–2218. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL https://www.usenix.org/conference/usenixsecurity21/presentation/lehmkuhl.

Ruixiao Li, Yu Ishimaki, and Hayato Yamana. Fully homomorphic encryption with table lookup for privacy-preserving smart grid. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 19–24, 2019. doi: 10.1109/SMARTCOMP.2019.00023.

Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. *Oblivious Neural Network Predictions via MiniONN Transformations*, pp. 619–631. Association for Computing Machinery, New York, NY, USA, 2017. ISBN 9781450349468. URL https://doi.org/10.1145/3133956.3134056.

Jeffrey L. McKinstry, Steven K. Esser, Rathinakumar Appuswamy, Deepika Bablani, John V. Arthur, Izzet B. Yildiz, and Dharmendra S. Modha. Discovering low-precision networks close to full-precision networks for efficient embedded inference, 2019.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. USENIX Association, August 2020a. ISBN 978-1-939133-17-5. URL https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi codebase. https://github.com/mc2-project/delphi, 2020b.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38, 2017. doi: 10.1109/SP.2017.12.

Renkun Ni, Hong min Chu, Oscar Castañeda, Ping yeh Chiang, Christoph Studer, and Tom Goldstein. Wrapnet: Neural net inference with ultra-low-resolution arithmetic, 2020.

Stefan Rass, Peter Schartner, and Monika Brodbeck. Private function evaluation by local two-party computation. *EURASIP Journal on Information Security*, 2015, 12 2015. doi: 10.1186/s13635-015-0025-9.

Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow2: Practical 2-party secure inference. *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, Oct 2020. doi: 10.1145/3372297.3417274. URL http://dx.doi.org/10.1145/3372297.3417274.

Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. Deepsecure: Scalable provably-secure deep learning, 2017.

SEAL. Microsoft SEAL (release 3.6). https://github.com/Microsoft/SEAL, November 2020. Microsoft Research, Redmond, WA.

J. Thaler, Mike Roberts, M. Mitzenmacher, and H. Pfister. Verifiable computation with massively parallel interactive proofs. *ArXiv*, abs/1202.1350, 2012.

McKenzie van der Hagen and Brandon Lucia. Practical encrypted computing for iot clients, 2021.

Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pp. 162–167, 1986. doi: 10.1109/SFCS.1986.25.

Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1lBj2VFPS.

Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. Non-vacuous generalization bounds at the imagenet scale: A pac-bayesian compression approach, 2019.