Private Quantised Neural Network Inference *

Hendrik Eerikson[1,2]✉[0009−0009−9298−074X], Sven Laur[1][0000−0002−9891−3347], and Pille Pullonen-Raudvere[2][0000−0002−3255−7001]

[1] University of Tartu, Institute of Computer Science, Tartu, Estonia
[2] Cybernetica AS, Information Security Research Institute, Tallinn, Estonia

Abstract. Neural networks based machine learning models are used in many classification problems yet privacy issues in model usage are mostly unresolved. When a model owner provides their model to other parties for inference, either the model owner has to share the parameters of the model or the model user must share their query and result with the model owner. Depending on the usage scenario, the user's query or query result could contain sensitive information that the model provider should not see. And on the other hand, the model parameters themselves could be a valuable business secret or leak private training data. In such cases, neural network inference must be performed in a privacy preserving manner, for example, using secure multi-party computation (MPC).
We propose protocols for the private inference of neural networks and evaluate them within the Sharemind MPC secure computation framework. Our protocols are based on the per-tensor quantisation scheme in PyTorch which maps floating point model parameters to 8-bit integer values. Quantisation improves the efficiency of our protocols and our protocols are interoperable with models trained using PyTorch. We extend constant round integer convolution and matrix multiplication protocols for quantised model weights, propose constant round protocols for re-quantising values and analyse the optimal secret share bit width for efficient re-quantisation. The resulting protocols are used to evaluate the VGG16 image classification network with an amortised throughput of 24.3 seconds per image classification in a LAN setting.

Keywords: Multi-party computation · privacy-preserving machine learning · neural networks · secure inference

1 Introduction

Privacy-preserving inference of neural networks can enable new data driven applications. Predictive models using neural networks are widely and successfully used for image classification, spam filtering, text translation, speech recognition and in other fields. But for applications where the query and its result are

* The Version of Record of this contribution is published in Availability, Reliability and Security. ARES 2026 EU Projects Symposium Workshops, and is available online at https://doi.org/10.1007/978-3-032-37218-5_27

particularly sensitive, neural networks have not made an impact. For example, for CSAM detection in messaging or cloud storage services as well as detection of rare medical conditions, existing implementations have triggered significant public privacy concerns [2, 13, 26].

To illustrate the security challenges, consider using neural networks for medical diagnosis as a service. The query and its result are obviously sensitive and should be seen only by the patient and their doctor. The model would be considered a medical device developed by some company at a great cost, meaning the model provider wants to keep it private. The model could also have been trained on sensitive data which could be extracted from the model parameters. Thus, the model parameters and query are secret data that belong to different parties and the goal of secure inference can be achieved with multi-party computation.

Multi-party computation (MPC) enables mutually distrusting parties to jointly compute on joint data without revealing secret information. The landscape of MPC is very diverse in terms of the security guarantees, the complexity of deployment, performance, the supported operations and programmability. All the possible approaches have their own benefits and drawbacks. For example, two-party protocols are the easiest to deploy but they require more complex cryptographic primitives and are generally less performant compared to three-party protocols with two honest parties. Actively secure protocols provide privacy and correctness guarantees against adversaries who can deviate from the protocol, yet they come with a performance penalty compared to similar passively secure protocols where the adversary must follow the protocol.

Although, an actively secure inference protocol can protect the correctness of model evaluation, it can not protect against an attack that is equivalent to one of the parties selecting a different input. Thus, active security is not sufficient to argue that the result is the correct evaluation of the correct model. In the medical diagnosis example, both the query and model provider are incentivised to provide the correct input and follow the protocol. If the secure computation is outsourced, similar incentives and organisational security measures must be in place to ensure correct evaluation of the protocol. However, if deviating from the protocol gives an advantage in leaking some private input, the protocol is not private. Privacy against passive adversaries is not sufficient when the data is as sensitive as, for example, medical records. Our protocol is private against an active adversary.

1.1 Our Contribution

We present honest-majority three-party protocols for evaluating neural networks where all parties are assumed to follow the protocol. Our protocol is private against an active adversary but ensures correctness only for a passive adversary. The privacy of the query, the model weights and the query result are protected. Our protocol does not require a preprocessing phase or correlated random inputs. We use the combination of input-private and secure components to obtain efficient secure protocols as outlined in [3].

We cover the evaluation of fully connected, convolution, ReLU and MaxPool layers. The core of the protocols is a new optimised matrix multiplication protocol and normalisation protocol for quantised values. We implement our protocols in the Sharemind MPC framework using its core three-party additive secret sharing scheme. Using the same secret sharing scheme enables the use of Sharemind MPC's large standard library and falling back on secure floating point computation for other activation functions. The protocols that we have developed form a library for privately evaluating common neural network architectures trained outside of MPC, for example using PyTorch.

We evaluate our implementation on the VGG16 [32] convolutional network for image classification. The VGG16 model has 138 million parameters, of which most are in the linear layers. As a reasonable mid range model, it has been used as a benchmark for many implementations (e.g. Falcon [33], XONN [30], FANNG-MPC [1], Orca [15], Piranha [34], PIGEON [12]).

The main goal in our protocol design is to minimise the communication complexity of the protocol without introducing costly cryptographic primitives such as pairings and homomorphic encryption. In this context, the main bottleneck is the memory footprint of the neural network parameters and its evaluation graph. This problem has been extensively studied in the machine-learning community [25, 35]. The resulting quantised neural network models use integer arithmetic that is more efficient and easier to implement as MPC protocols. We establish the optimal bit width of the secret sharing to prevent overflow and enable efficient normalisation of shares while minimising communication in the protocols. Additionally, we precisely profile the execution of our protocol in different network settings to gain a clear view of where performance bottlenecks lie and which optimisation strategies should be further pursued.

2 Preliminaries

2.1 Secure Multi-Party Computation

The goal of secure multi-party computation (MPC) is to enable distrusting parties to compute on their joint data while ensuring the privacy of the data. An MPC protocol is said to have active security if it ensures security with respect to any adversarial behaviour. Conversely, the protocol has passive security if security is ensured for participants that follow the protocol. In many cases, a protocol can ensure privacy against an active adversary while only ensuring correctness (and, therefore, security) against a passive adversary [11]. There are various concrete technologies used to build MPC protocols such as secret sharing, function secret sharing, garbled circuits and (fully) homomorphic encryption. In addition to these, secure hardware can be used to give similar guarantees. Each of these approaches has their own strengths and weaknesses, for further discussion see [21, 31].

We focus on reducing the amount of communication needed to compute private inference with three parties and additive secret sharing. Often, secure

computation protocols are split into a precomputation phase that generates correlated randomness and an online phase that uses the inputs of the parties and the correlated randomness to compute the results. Precomputation can be used to make the online phase very efficient but the combined complexity is often ignored. Efficient precomputation can be devised if the shape of the computation is known whereas optimizing more general protocol execution is difficult. In this work, we consider protocols without precomputation.

2.2 Three-Party Additive Secret Sharing

Our protocols are based on additive secret sharing over rings $\mathbb{Z}_{2^\ell}$. We denote a secret sharing of $x \in \mathbb{Z}_{2^\ell}$ as $[\![x]\!]_\ell = \{x_1, x_2, x_3\}$ where $x_1 + x_2 + x_3 = x \mod 2^\ell$. For brevity, the subscript ℓ is sometimes omitted. The computing parties are denoted as P_1, P_2 and P_3, P_{i+1} and P_{i-1} denote the next and previous parties respectively (the index rolls over). Similarly, we denote the share of $[\![x]\!]$ that belongs to party P_i as $[\![x]\!]^i$. Matrices are written using capital letters, e.g. X, and their elements in the ith row and jth column are denoted with the respective lowercase letter x_{ij}.

To secret share a value $x \in \mathbb{Z}_{2^\ell}$, uniformly sample x_1 and x_2 from $\mathbb{Z}_{2^\ell}$, set $x_3 \leftarrow x - x_1 - x_2 \mod 2^\ell$, give the share x_i to P_i. To reconstruct a secret shared value, receive the shares from the three parties and find the sum modulo 2^ℓ.

This secret sharing scheme is *perfectly private* with a threshold of 3-out-of-3, meaning that any set of two or fewer shares leaks no information about the underlying value and it can be completely reconstructed from three shares. Computing linear combinations of secret shared values requires no communication:

$$[\![x + y]\!] = \{x_1 + y_1, x_2 + y_2, x_3 + y_3\}$$
$$c \cdot [\![x]\!] = \{c \cdot x_1, c \cdot x_2, c \cdot x_3\}$$

Resharing is an operation that re-randomises the shares while keeping the same value. To reshare $[\![x]\!]$, each party P_i samples a random value r_i and sends it to the next party, the new secret sharing of x is then

$$\mathsf{reshare}([\![x]\!]) = \{x_1 + r_1 - r_3, x_2 + r_2 - r_1, x_3 + r_3 - r_2\}$$

As a common optimisation technique, each pair of parties can agree on a pseudo-random number generator seed beforehand so that both parties P_i and P_{i+1} can uniformly sample the same value instead of sending the sampled value, making the resharing protocol non-interactive.

Sometimes it is useful to share a secret value between just two parties such that $[\![x]\!]_\ell = \{x_1, x_2, 0\}$ and $x = x_1 + x_2 \mod 2^\ell$. A single-round interactive protocol for achieving this is denoted as $\mathsf{reshareToTwo}([\![x]\!])$.

Multiplication of shared values arises naturally from the distributive property of multiplication, see Alg. 1. This protocol is a modification of the multiplication protocol proposed in [5] that also reshares the final output z before returning it as output. Removing the final reshare was discussed in [3]. While the final

Algorithm 1 Multiplication Protocol [5]

Require: $[\![x]\!]$ and $[\![y]\!]$ are additively shared over $\mathbb{Z}_{2^\ell}$
Ensure: $[\![z]\!] = [\![x \cdot y]\!]$
$[\![x']\!] \leftarrow \mathsf{reshare}([\![x]\!])$
$[\![y']\!] \leftarrow \mathsf{reshare}([\![y]\!])$
P_i sends $[\![x']\!]^i$ to P_{i+1} and $[\![y']\!]^i$ to P_{i-1}
$[\![z]\!]^i \leftarrow [\![x']\!]^i[\![y']\!]^i + [\![x']\!]^{i+1}[\![y']\!]^i + [\![x']\!]^{i+1}[\![y']\!]^{i-1}$
return $[\![z]\!]$

resharing can be important to ensure that the outputs are simulatable, it is not necessary to ensure privacy. It is possible to omit the final resharing when combining many multiplications, only resharing the final output of the computation. Hence, we can use this reduced multiplication protocol as a component in matrix multiplication. Note that while the protocol preserves security against a passive adversary, it is perfectly private against the active adversary [28]. The latter means that while an active adversary may make the output of the protocol incorrect, it does not learn information about the private inputs from this protocol.

Note that after the communication steps, the final part of the multiplication protocol is local computation. We use this property in the convolution and matrix multiplication protocols (Alg. 2) to reduce the required communication. The same multiplication protocol can be used for computing the binary AND operation in which case $x \in \mathbb{Z}_2$. The protocol for binary OR can also be constructed similarly.

2.3 Convolutional Neural Networks

Convolutional neural networks are used across a wide range of computer vision application domains, including image classification, medical imaging, and facial recognition. We use the VGG16 image classification model [32] as our motivating example. VGG16 consists of 13 convolution layers, 3 fully connected linear layers, 5 MaxPooling layers and ReLU activation functions. The model has 138 million parameters, most of which are in the linear layers. While VGG16 has been superseded by newer model architectures, models based on the same building blocks of convolution, linear and pooling layers with ReLU activations are still relevant for many computer vision tasks. In plaintext inference of VGG16 [32], convolution layers and fully connected layers form the majority of the computation costs. Thus, it is important that these layers are efficiently evaluated. Similarly, optimising the secure computation of convolution and fully connected layers is critical for efficient secure inference.

2.4 Quantisation

Quantisation reduces the memory footprint of neural network models by reducing the size and precision of model parameters. Generally, neural networks are trained with 32-bit or 16-bit floating point parameters but after training, for

precise inference, less precise discretised parameters suffice. Many schemes for discretising the model parameters exist varying in the size of the parameters, accuracy trade-off and quantisation algorithm [10,18,20]. In plaintext inference, quantisation is used to fit very large models into the limited size but high-speed memory of GPUs. In private inference, it reduces the communication required for inference and simplifies the evaluation of the neural network's layers. While the non-quantised model requires floating point arithmetic, which is expensive in MPC, the quantised model requires mostly integer arithmetic. Note that in weight-only quantisation schemes, such as AWQ or GPTQ, quantised weights are converted back to floating point values for computing layer activations, thus these schemes are not viable for secret sharing.

In this work, we use per-tensor post-training 8-bit integer quantisation as is supported by PyTorch[3]. For a real number value x, the quantised value is

$$\bar{x} = \left\lfloor \frac{x}{\Delta} - z \right\rceil$$

where Δ is a scaling factor and z is the quantisation zero point, both of which are determined per each tensor in the computation. For 8-bit quantisation, the scaling factor and zero point are such that for all values x that an element of the tensor can take, the quantised value $\bar{x}$ is in the range $[0, 255]$ for unsigned or in $[-128, 127]$ for signed quantisation. The bit-width of the quantised values could be larger or smaller to either improve inference accuracy or reduce the memory footprint. The 8-bit quantisation was chosen as it has wide support among PyTorch modules, meaning that most neural networks in PyTorch can be automatically quantised to 8 bits.

2.5 Private Inference

In privacy-preserving machine learning (PPML), we are interested in ensuring the privacy of all or some of the following: the training data, the trained model, the input to an inference query and the query result. In private inference, the query of the client is protected and the client should not learn the model parameters. If implemented with MPC, then either the model is also protected or known to the computing parties. In this work, we are implementing a scenario where both the query and the model parameters are secret shared. A recent overview of PPML approaches can be found in [7, 14]. For a benchmark, we consider VGG16 image classification model, which is a commonly used large benchmark for PPML. For example, it is supported by Falcon [33], XONN [30], FANNG-MPC [1], Orca [15], Piranha [34], PIGEON [12].

Using floating-point arithmetic in PPML is improving [29], but using fixed-point arithmetic is more common [1,33]. Approaches, such as [9,15,16,22], focus on using quantised models. Whereas, [34] uses approximations of the non-linear computations which reduces the accuracy of the results and [30] uses binary neural networks.

[3] `https://pytorch.org/docs/stable/quantization.html`

Significant work in PPML has been dedicated to optimizing the matrix multiplication and the inner product protocols. ABY3 [23] uses replicated secret sharing which allows to compute the additive shares of the inner product locally, whereas resharing is necessary to get the replicated shares of the output. However, the input privacy idea allows us to reuse the share throughout the matrix multiplication protocol and we do not need to reshare the final output. ASTRA [6], Blaze [27] and Swift [17] design secret sharing schemes to allow efficient operations for machine learning and also extend the ideas of ABY3. Note that these protocols rely on a precomputation phase.

Not all PPML frameworks and protocols consider the model parameters as secret inputs. Opening the model parameters to the computing parties results in protocols with much lower communication. However, if the model is not secret, then the query party could compute inference locally, making PPML unnecessary. If the model is kept secret from the query party but the computing parties are allowed to see it, then the deployment requires finding computing parties that can be trusted with the model but are also trusted to not collude to reveal the query. A PPML framework that ensures the secrecy of both the model and the query against an active adversary is most fitting for real-life deployment.

3 New Protocols for Private Inference

We propose secure computation protocols for evaluating quantised neural network models such that the model parameters, queries and results are secret shared. The model is secret shared among the MPC computing parties. An input party secret shares their query, sends shares to the computing parties and receives shares of the output.

3.1 Fully Connected and Convolution Layers

Given real number matrix X with elements x_{ij}, a matrix consisting of quantised elements $\overline{x_{ij}}$ is such that

$$x_{ij} = \Delta_x \cdot (\overline{x_{ij}} - z_x)$$

where Δ_x is the scaling factor and z_x is the zero-point specific to the matrix X. The scaling factor and zero-point are determined during neural network quantisation for each tensor in the network evaluation and are not considered secret inputs. It is possible to keep them confidential, but it introduces a computational overhead. Moreover, the potential information leakage is minor compared to leakage from the outputs of the neural network itself.

Let the matrix product $P = X \cdot Y$ have elements p_{ij}. The real matrix products can be computed in terms of the quantised matrices as such:

$$\begin{aligned} p_{ij} &= \sum_k x_{ik} \cdot y_{kj} = \sum_k \Delta_x(\overline{x_{ik}} - z_x) \cdot \Delta_y(\overline{y_{kj}} - z_y) = \\ &= \Delta_x \Delta_y \cdot \sum_k (\overline{x_{ik}} - z_x)(\overline{y_{kj}} - z_y) \end{aligned}$$

Algorithm 2 Matrix Multiplication Protocol

Require: $[\![X]\!]$ and $[\![Y]\!]$ are matrices of additive shares over $\mathbb{Z}_{2^\ell}$ with dimensions (M, K) and (K, N) respectively.
Ensure: $[\![W]\!] = [\![XY]\!]$
  $[\![X']\!] \leftarrow \mathsf{reshare}([\![X]\!])$
  $[\![Y']\!] \leftarrow \mathsf{reshare}([\![Y]\!])$
  each party P_i sends their share of $[\![X']\!]$ to party P_{i+1}
  each party P_i sends their share of $[\![Y']\!]$ to party P_{i-1}
  for $m \in \{1, \ldots, M\}$ **do**
    for $n \in \{1, \ldots, N\}$ **do**
      for each party P_i:
      $[\![w_{mn}]\!]^i \leftarrow \sum_{k=1}^{K} [\![x'_{mk}]\!]^i \cdot [\![y'_{kn}]\!]^i + [\![x'_{mk}]\!]^{i+1} \cdot [\![y'_{kn}]\!]^i + [\![x'_{mk}]\!]^{i+1} \cdot [\![y'_{kn}]\!]^{i-1}$
    end for
  end for
  return $[\![W]\!] = ([\![w_{mn}]\!])$

and the quantised matrix product is computed as such:

$$\overline{p_{ij}} = \left\lfloor \frac{\Delta_x \Delta_y}{\Delta_p} \cdot \Big(\sum_k (\overline{x_{ik}} - z_x)(\overline{y_{ki}} - z_y) \Big) + z_p \right\rceil. \tag{1}$$

In the default PyTorch quantisation scheme, input and activation tensor are quantised to 8-bit unsigned integers, model weight tensors are quantised to 8-bit signed integers, the zero-point is an 8-bit unsigned integer and the scale is a 32-bit floating point value. When evaluating equation 1, the quantised matrix element offset by the zero-point $(\overline{x_{ik}} - z_x)$, is stored as a 16-bit integer and the product and sum are stored as 32-bit integers.

Our secure protocol for quantised matrix multiplication must compute Equation 1. Thus, the protocol steps are integer matrix multiplication, multiplication by a scalar fraction and clamping the result to 8 bits. The integer matrix multiplication uses the secret shared multiplication protocol from Alg. 1, but since in matrix multiplication each matrix element appears as an operand in many scalar multiplications we can reuse the communication from earlier scalar multiplications. In fact, we can communicate the shares of the input matrices beforehand and the remainder of the integer matrix multiplication is only local operations. The optimised protocol is detailed in Alg. 2. Note that we optimise the full matrix multiplication not just the inner products within the matrix multiplication.

Suppose our input matrices have sizes (M, K) and (K, N), then the unoptimised matrix multiplication consists of $M \cdot K \cdot N$ multiplications, each requiring sending two shares per party. As a result of reusing the sent shares between the scalar multiplications, only $K(M + N)$ shares must be sent per party for the entire integer matrix multiplication. Note that this is the same as the online communication required by the Beaver triple based multiplication using matrix triples [8, 24]. However, Beaver triple based multiplication also requires communication in the precomputation phase and the matrix shapes must be known in advance. We focus on fully-online multiplication to accommodate variable size matrices during runtime.

Algorithm 3 Secure Quantised Matrix Multiplication Protocol

Require: $[\![X]\!]$ and $[\![Y]\!]$ are matrices of additive shares of $\mathbb{Z}_{2^{32}}$, $\Delta_x, \Delta_y, \Delta_w, z_x, z_y, z_w$ are public constants.
Ensure: $[\![W]\!]$ is the quantised matrix multiplication of X and Y.

$[\![X']\!] \leftarrow [\![X]\!] - z_x$
$[\![Y']\!] \leftarrow [\![Y]\!] - z_y$
$[\![A]\!] \leftarrow \mathsf{matmul}([\![X']\!], [\![Y']\!])$ ▷ Alg. 2
$s \leftarrow \frac{\Delta_x \Delta_y}{\Delta_w}$
for $[\![a_{ij}]\!] \in [\![A]\!]$ **do**
  $[\![a'_{ij}]\!] \leftarrow \mathsf{reshareToTwo}(a_{ij})$
  $[\![c]\!] \leftarrow \mathsf{carry}([\![a'_{ij}]\!])$ ▷ Alg. 4
  $[\![w'_{ij}]\!] \leftarrow [\![\lfloor s \cdot a'_{ij} \rceil]\!] - \lfloor s \cdot 2^{32} \cdot [\![c]\!] \rceil + z_w$
  $[\![w_{ij}]\!] \leftarrow \mathsf{clamp}([\![w'_{ij}]\!])$ ▷ Alg. 5
end for
return $[\![W]\!] = (w_{ij})$

Intuitively, this optimised matrix multiplication protocol is private as no party sees any more information than they would if the matrix multiplication were performed using the scalar multiplication protocol in Alg. 1. On the other hand, significant reuse of the shares might introduce strong dependencies between the different shares that we get as an output and introduce a security risk if published [3]. Formally, the multiplication protocol is a composition of input private parts that makes it input private itself [3]. Hence, using the blueprint for secure protocols from [3], we can use this multiplication to build secure protocols if the outputs of the matrix multiplication are consumed by other secure computation protocols that ensure that the low randomness in the outputs is hidden when they are used. This is indeed the case for the full neural network evaluation that uses this multiplication protocol.

Using the integer multiplication as a sub-protocol, we construct the protocol for the quantised matrix multiplication (Equation 1) as Alg. 3. The final part of the protocol normalises the result of the integer matrix multiplication to the range of the quantised output. The protocols required for this are described in the following subsection.

The convolution protocol is optimised similarly to matrix multiplication by sending the shares of the image and the kernel before performing the rest of the protocol locally. Thus, the communication pattern of the convolution protocol is the same as the matrix multiplication protocol and the only difference is in the local computation. As with the matrix multiplication protocol, the privacy of the convolution protocol follows from the input-privacy argument.

3.2 Normalisation of Quantised Shares

After the integer matrix multiplication step in Alg. 3, the elements of the resulting matrix are not in the 8-bit interval of the quantised output. Thus, the result needs to be scaled by the scaling factor and clamped to the range of 8-bit signed integers. The scaling factor $s = \frac{\Delta_x \Delta_y}{\Delta_w}$ is represented as a public 32-bit floating point value.

To scale a secret shared integer by a floating point value, it is not sufficient to simply multiply each share with the float. To simplify the explanation, let $[\![x]\!] = \{x_1, x_2\}$ such that $x_1 + x_2 = x \mod N$, then $x_1 + x_2 = x - c \cdot N$ where $c \in \{0, 1\}$. By multiplying the shares with s we get $sx_1 + sx_2 = sx + scN$, since scN rounded to the nearest integer is not necessarily divisible by N, $sx_1 + sx_2 \neq sx \mod N$. To get the correct scaled value $[\![y]\!] = [\![\lfloor s \cdot x \rceil]\!]$, we have to correct for the carry c by, for example, setting $y_1 = \lfloor sx_1 \rceil - \lfloor scN \rceil \mod N$ and $y_2 = \lfloor sx_1 \rceil \mod N$. Since the carry c leaks information about x, we must compute the carry and perform the correction step under secret sharing.

One option for finding the carry is through bit decomposition using a process similar to Alg. 6 but such an approach is very costly. Instead, we can take advantage of the fact that we use an integer ring for secret sharing that is much larger than the plaintext values. If we look at the binary representation of the value x and its shares x_1 and x_2, then c is the final carry bit of the addition of x_1 and x_2. The carry c can be computed as such:

$$c = (x_1[-1] \cdot x_2[-1]) + (c_{in} \cdot (x_1[-1] + x_2[-1]))$$

where the suffix $[-1]$ indicates the highest bit of an integer and c_{in} is the carry from adding the second to last bits. But, if $x < N/2$, the highest bit of x is 0 meaning that c does not depend on the incoming carry. This means that we can compute c without having to ripple the carry bits through the entire addition of x_1 and x_2. The resulting protocol is detailed in Alg. 4.

Algorithm 4 Carry Bit Protocol

Require: $[\![x]\!] = \{0, x_2, x_3\}$, $x \in \mathbb{Z}_N$ and $x < N/2$.
Ensure: $[\![c]\!]$ such that $x_2 + x_3 = x - c \cdot N$
  $[\![a]\!]_2 \leftarrow \{0, \mathsf{highestBit}(x_2), 0\}$
  $[\![b]\!]_2 \leftarrow \{0, 0, \mathsf{highestBit}(x_3)\}$
  $[\![c]\!] \leftarrow \mathsf{shareconv}([\![a]\!]_2 \vee [\![b]\!]_2)$
  return $[\![c]\!]$

The sub-protocol $\mathsf{shareconv}([\![x]\!]_2)$ takes a value secret shared in a smaller ring and returns the same value secret shared in a larger ring [5].

In the clamping protocol (Alg. 5), we need to decide whether the input is greater than 255, less than 0 or in-between the two and choose the appropriate output. Notice that clamping is not equivalent to taking the bottom 8 bits of the input. In the protocol, we convert secret shares of x over $\mathbb{Z}_{2^\ell}$ to binary shares $[\![x_1]\!]_2, \ldots, [\![x_\ell]\!]_2$ where

$$x = \sum_{i=1}^{\ell} 2^{i-1} x_i.$$

We call this conversion bit extraction (Alg. 6). In this representation, we can compute whether x is already in the 8-bit range or if it is over or under it.
In Alg. 5, $[\![c]\!]_2 = \mathsf{foldOR}([\![x_1]\!]_2, \ldots, [\![x_n]\!]_2)$ is a sub-protocol call, which computes

$$c = \bigvee_{i=1}^{n} x_i$$

Algorithm 5 Secure Clamp to Interval $[0, 255]$ Protocol

Require: $[\![x]\!]$ is a two's complement signed integer additively shared over $\mathbb{Z}_{2^{32}}$.

Ensure: $[\![y]\!] = \begin{cases} [\![x]\!] \text{ if } 0 \leq x \leq 255 \\ [\![255]\!] \text{ if } x > 255 \\ [\![0]\!] \text{ if } x < 0 \end{cases}$

$[\![x_1]\!]_2, \ldots, [\![x_{32}]\!]_2 \leftarrow \mathsf{bitExtract}([\![x]\!])$ ▷ Alg. 6
$[\![c]\!]_2 \leftarrow \mathsf{foldOR}([\![x_9]\!]_2, \ldots, [\![x_{31}]\!]_2)$
$[\![a]\!] \leftarrow \mathsf{choose}([\![c]\!]_2, [\![255]\!], [\![x]\!])$
$[\![y]\!] \leftarrow \mathsf{choose}([\![x_{32}]\!]_2, [\![0]\!], [\![a]\!])$
return $[\![y]\!]$

through a folding algorithm, giving a protocol with $O(\log_2(n))$ communication depth. The sub-protocol $\mathsf{choose}([\![b]\!]_2, [\![x]\!], [\![y]\!])$ outputs the secret sharing of either x or y based on the value of the control bit b. Both of these sub-protocols are introduced in [5].

The bit extraction protocol (Alg. 6) works by sharing the input between two parties and using those two shares as if they were binary secret shares, which are then added using a binary adder circuit. The binary adder circuit can be implemented using secret shared operations over $\mathbb{Z}_2$. For brevity, we detail a simple binary addition protocol based on the ripple-carry adder in Alg. 7 but use a more efficient implementation based on the Kogge-Stone adder [19] in the actual protocol. The protocol based on the Kogge-Stone adder trades increased communication for fewer rounds and simpler local computation.

Algorithm 6 Bit Extraction Protocol [19]

Require: $[\![x]\!]$ is additively shared over $\mathbb{Z}_{2^\ell}$
Ensure: $[\![y_1]\!]_2, \ldots, [\![y_\ell]\!]_2$ such that $x = \sum_{i=0}^{\ell} 2^{i-1} \cdot y_i$
$[\![x']\!] \leftarrow \mathsf{reshareToTwo}([\![x]\!])$
Let $[\![x']\!] = \{0, a, b\}$
$[\![a_i]\!]_2 \leftarrow \{0, a[i], 0\}$ for $i \in 1, \ldots, \ell$
$[\![b_i]\!]_2 \leftarrow \{0, 0, b[i]\}$ for $i \in 1, \ldots, \ell$
$([\![y_i]\!]_2)_{i=1}^{\ell} \leftarrow \mathsf{binAdd}([\![a_1]\!], \ldots [\![a_\ell]\!], [\![b_1]\!], \ldots, [\![b_\ell]\!])$
return $[\![y_1]\!], \ldots, [\![y_\ell]\!]$

Algorithm 7 Ripple-Carry Adder Protocol

Require: $[\![x_1]\!]_2, \ldots, [\![x_\ell]\!]_2$ and $[\![y_1]\!]_2, \ldots, [\![y_\ell]\!]_2$ such that $x = \sum_{i=1}^{\ell} 2^{i-1} \cdot x_i$ and $y = \sum_{i=1}^{\ell} 2^{i-1} \cdot y_i$
Ensure: $[\![z_1]\!]_2, \ldots, [\![z_\ell]\!]_2$ s.t. $\sum_{i=1}^{\ell} 2^{i-1} \cdot z_i = x + y$
$[\![c_0]\!] \leftarrow [\![0]\!]_2$
for $i \in 1, \ldots, \ell$ **do**
  $[\![z_i]\!] \leftarrow [\![x_i]\!] \oplus [\![y_i]\!] \oplus [\![c_{i-1}]\!]$
  $[\![c_i]\!] \leftarrow ([\![x_i]\!] \wedge [\![y_i]\!]) \oplus ([\![c_{i-1}]\!] \wedge ([\![x_i]\!] \oplus [\![y_i]\!]))$
end for
return $[\![z_1]\!], \ldots, [\![z_\ell]\!]$

After scaling and clamping, all but the last layer of VGG16 are followed by the ReLU activation function which simply outputs 0 if the input x is negative and x otherwise. In our quantised scheme, we must compare x against the zero point of the corresponding tensor instead. Because after clamping, x is necessarily an 8-bit value, we can truncate the shares of x and perform ReLU on $\mathbb{Z}_{2^8}$. Truncation of shares is a local only operation. The protocol for ReLU is given in Alg. 8, it uses the integer comparison protocol from [4].

Algorithm 8 Quantised ReLU Protocol

Require: $[\![x]\!]$ and zero point z

Ensure: $[\![y]\!] = \begin{cases} [\![z]\!] \text{ if } x < z \\ [\![x]\!] \text{ otherwise} \end{cases}$

$[\![c]\!] \leftarrow \mathsf{lessThan}([\![x]\!], z)$

return $\mathsf{choose}([\![c]\!], [\![z]\!], [\![x]\!])$

The protocol for the 2×2 MaxPooling layer in VGG16 is implemented similarly to the ReLU protocol and also performed on shares in $\mathbb{Z}_{2^8}$. After ReLU and MaxPooling, the shares are converted back to $\mathbb{Z}_{2^{32}}$ using the share conversion protocol from [5].

The communication cost in terms of total bits sent between all parties over all rounds and the number of communication rounds are given in Table 1. In low bandwidth network settings, the amount of communicated bits is the constraining factor, in high latency settings, the number of rounds is the constraining factor. Note that for bit extraction and clamp protocols, we give the communication amount for the implementation that has been machine optimised by the PDSL [19] compiler. The compiler's optimisations remove redundant messages between parties, reducing communication slightly.

Table 1. Communication amount and round count of selected protocols.

Protocol	Input	Comm. bits	Rounds
Mult. (Alg. 1)	$x, y \in \mathbb{Z}_{2^\ell}$	6ℓ	1
Matrix mult. (Alg. 2)	$X \in \mathbb{Z}_{2^\ell}^{M \times K}$ $Y \in \mathbb{Z}_{2^\ell}^{K \times N}$	$6\ell \cdot K(M+N)$	1
Carry bit (Alg. 4)	$x \in \mathbb{Z}_{2^\ell}$	$6 + \ell$	2
Bin. add (Alg. 7)	$x, y \in \mathbb{Z}_{2^\ell}$	12ℓ	$\ell - 1$
Bit extract (Alg. 6)	$x \in \mathbb{Z}_{2^{32}}$	339	32
Bit extract [19]	$x \in \mathbb{Z}_{2^{32}}$	1120	7
Clamp (Alg. 5)	$x \in \mathbb{Z}_{2^{32}}$	1646	15

3.3 Analysis of Required Share Size

In the implementation of our PPML framework, we use 32-bit shares to store the 8-bit quantised weights and neuron activation values of the network. The

larger share size means that the results of arithmetic operations on shared values will not overflow and we can omit an expensive share extension protocol. As a drawback, due to the larger shares, the communication amount and round count of our protocols is larger than it would be with smaller shares.

The analysis to substantiate the choice of 32-bit shares follows the steps of the protocol. The input values and weights of a quantised neural network layer are 8-bit integers. Assuming the zero-point of the input matrices is also an 8-bit value, the result of subtracting the zero-point from a matrix can fit into 9 bits. Note that in the PyTorch plaintext implementation of the quantisation scheme, this intermediate result is 16-bit. Within the matrix multiplication or convolution, the result of a scalar multiplication of 9-bit values is 18-bit. A number of these products is summed together, depending on the matrix dimensions or convolution kernel size. With 32-bit shares, in the worst case, the summation will overflow with 2^{14} addends. In VGG16, the largest number of addends in a single layer is 2^{12}, thus the highest bit of the sum will necessarily be 0, as required for the optimised carry bit protocol in Alg. 4.

4 Performance Evaluation

We implement our protocols within the Sharemind MPC framework. The carry, clamp, ReLU and MaxPool protocols are implemented using PDSL, a domain specific language for writing MPC protocols [19]. The compilation process of PDSL performs automatic rewrites of the computation graph to optimise arithmetic expressions. The compiler also schedules expression evaluation such that a minimal number of communication rounds is required. A static analyser is used on the optimised computation graph to automatically prove privacy against an active adversary [28]. The code generated from the PDSL protocol implementations is single-threaded LLVM IR.

The rest of the protocols are implemented in C++. The local computation part of matrix multiplication and convolution protocols is deferred to torchlib, the C++ backend library of PyTorch. The torchlib library performs matrix multiplication and convolution on multiple CPU threads. The library is capable of performing the computations on GPUs and other accelerators but we did not have the specialised hardware to utilise this.

4.1 Analysis and Comparison to Related Work

We perform our measurements on a cluster of three servers, each has two Intel Xeon E5-2640 v3 2.6 GHz CPUs, 128 GB of system memory and 10 Gbit/s dedicated network links with each of the other servers. We use traffic shaping to simulate network conditions with lower bandwidth and higher latency. We set bandwidth to 10 Gbit/s and RTT (round-trip time) to 2 ms to simulate a deployment with dedicated network links between computing parties and a network with 1 Gbit/s bandwidth and 10 ms RTT to simulate separate data

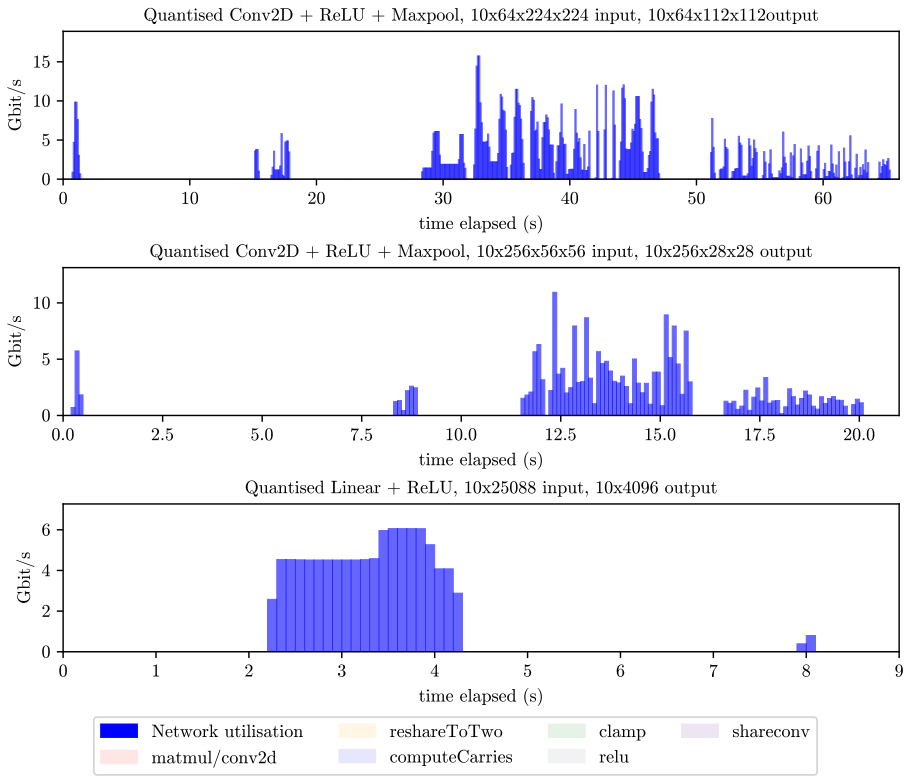


Fig. 1. Performance measurement of private inference of selected layers in the VGG16 model, 10 Gbit/s bandwidth, 2 ms RTT.

centres within the same region. Note that RTT includes the network latency in both directions.

In Fig 1, the private inference of three layers of the VGG16 model in a LAN setting is visualised. The coloured areas of the time series indicate phases of the computation and the blue bars indicate communication between the computing parties in Gbit/s. Deeper layers within the network have more parameters but smaller output sizes and thus lower overall computation time. In deeper layers of the network, the integer convolution protocol or integer matrix multiplication protocol take up a larger part of the computation time with low network utilisation, indicating an opportunity for optimisation using parallelisation and hardware acceleration.

Note that in Fig 1, network utilisation is measured in aggregate over all three parties, thus the theoretical maximum utilisation is 30 Gbit/s. In all layers, the protocol execution never reaches this theoretical maximum, indicating that in the LAN setting, the bottleneck lies in the local computation steps.

The total private inference time of the quantised VGG16 model in three different network conditions is given in Table 2. We measured the inference time

Table 2. Total inference time of VGG16 in various network conditions.

Setting	Inference time (seconds)			Amortised inference time
	batch=1	batch=10	batch=20	
10 Gbit/s 2 ms RTT	84.0	259.6	485.4	24.3
1 Gbit/s 2 ms RTT	103.7	422.7	831.5	41.6
1 Gbit/s 10 ms RTT	108.4	431.1	833.2	41.7

for batches of 1, 10 and 20 images, the input image size is $3 \times 224 \times 224$. In the LAN setting, the amortised throughput of inference is 24.3 seconds per image and the time to first result is 84 seconds. The total communication between all three parties is 7.1 GB for inferencing one image.

While multiple papers benchmark private inference using the VGG16 model, the results are unfortunately not directly comparable. The FANNG-MPC framework performs VGG16 inference in 534 seconds [1] but this result does not include preprocessing time. The Piranha platform reports VGG16 inference in 0.469 seconds with 88.39 GB of communication [34], this result is inconsistent with the reported network setting. The PIGEON framework performs VGG16 inference on a batch of 192 CIFAR-10 images in 0.92 seconds and on ImageNet images in 25.10 seconds [12]. It is unclear from the reported results how the different input image sizes affect the model architecture as the VGG16 model takes an input of size $3 \times 224 \times 224$. The PIGEON framework benchmarking results also claim to require 10 MB of total communication for evaluating the linear layers of VGG16 which contain 123 million parameters. The Falcon protocol implementation [33] computes the inference of a 32×32 image with the VGG16 model in 0.79 seconds (1.4 seconds reported in [34]). For a direct comparison of protocols, equivalent implementations and testing scenarios are needed.

5 Conclusion

We showed that the amortised running time of private inference of a deep convolutional neural network with private input, weights and output is 24 seconds in an ideal network setting. In a high bandwidth setting, further optimisation using parallelisation of local computations is substantiated. In low bandwidth settings, only a large reduction in the communication of the underlying protocols would improve performance significantly. While the presented protocols are applied to an image classification model, they are mostly applicable to other neural network architectures as well. Our chosen security setting with three parties, honest majority and privacy against an active adversary is well suited for a real world deployment. Reducing the number of computing parties to two or protecting against a dishonest majority would require much more expensive protocols that are based on asymmetric encryption. Adding a correctness guarantee against

active adversaries would increase the communication of the protocols while not making a significant impact in terms of real-world security.

Acknowledgments. The research was supported by the European Union through the European Defence Fund (EDF) under Grant Agreement No. 101121405, the Office of Naval Research Global (ONRG) under grant number N62909-23-1-2046, the Estonian Research Council through PRG1844, and the Estonian Ministry of Education and Research, through the Estonian Centre of Excellence in AI (EXAI), grant number TK213.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Aaraj, N., Aly, A., Güneysu, T., Marcolla, C., Mono, J., Paludo, R., Santos-González, I., Scholz, M., Soria-Vazquez, E., Sucasas, V., Suresh, A.: FANNG-MPC: framework for artificial neural networks and generic MPC. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2025**(1), 1–36 (2025). https://doi.org/10.46586/TCHES.V2025.I1.1-36, https://doi.org/10.46586/tches.v2025.i1.1-36
2. Belanger, A.: Apple hit with $1.2b lawsuit after killing controversial csam-detecting tool. https://arstechnica.com/tech-policy/2024/12/thousands-of-child-sex-abuse-victims-sue-apple-for-lax-csam-reporting/ (2024), ars Technica, Accessed: 2025-09-24
3. Bogdanov, D., Laud, P., Laur, S., Pullonen, P.: From Input Private to Universally Composable Secure Multi-party Computation Primitives. In: IEEE 27th Computer Security Foundations Symposium, CSF 2014. pp. 184–198. IEEE Computer Society (2014). https://doi.org/10.1109/CSF.2014.21, https://doi.org/10.1109/CSF.2014.21
4. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: Jajodia, S., Lopez, J. (eds.) Computer Security - ESORICS 2008. pp. 192–206. Springer Berlin Heidelberg, Berlin, Heidelberg (2008)
5. Bogdanov, D., Niitsoo, M., Toft, T., Willemson, J.: High-performance secure multi-party computation for data mining applications. International Journal of Information Security **11**(6), 403–418 (2012), http://dx.doi.org/10.1007/s10207-012-0177-2
6. Chaudhari, H., Choudhury, A., Patra, A., Suresh, A.: ASTRA: high throughput 3pc over rings with application to secure prediction. In: Sion, R., Papamanthou, C. (eds.) ACM SIGSAC Conference on Cloud Computing Security Workshop, CCSW (CCS) 2019. pp. 81–92. ACM (2019). https://doi.org/10.1145/3338466.3358922, https://doi.org/10.1145/3338466.3358922
7. Chen, C., Wei, L., Xie, J., Shi, Y.: Privacy-preserving machine learning based on cryptography: A survey. ACM Trans. Knowl. Discov. Data **19**(4) (May 2025). https://doi.org/10.1145/3729234, https://doi-org.ezproxy.utlib.ut.ee/10.1145/3729234
8. Chen, H., Kim, M., Razenshteyn, I.P., Rotaru, D., Song, Y., Wagh, S.: Maliciously secure matrix multiplication with applications to private deep learning. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology - ASIACRYPT 2020. LNCS, vol. 12493, pp. 31–59. Springer (2020). https://doi.org/10.1007/978-3-030-64840-4_2, https://doi.org/10.1007/978-3-030-64840-4_2

9. Dalskov, A.P.K., Escudero, D., Keller, M.: Secure evaluation of quantized neural networks. Proc. Priv. Enhancing Technol. **2020**(4), 355–375 (2020). https://doi.org/10.2478/POPETS-2020-0077, https://doi.org/10.2478/popets-2020-0077
10. Frantar, E., Ashkboos, S., Hoefler, T., Alistarh, D.: Gptq: Accurate post-training quantization for generative pre-trained transformers (2023), https://arxiv.org/abs/2210.17323
11. Genkin, D., Ishai, Y., Prabhakaran, M., Sahai, A., Tromer, E.: Circuits resilient to additive attacks with applications to secure computation. In: Shmoys, D.B. (ed.) Symposium on Theory of Computing, STOC 2014. pp. 495–504. ACM (2014). https://doi.org/10.1145/2591796.2591861, https://doi.org/10.1145/2591796.2591861
12. Harth-Kitzerow, C., Wang, Y., Rajat, R., Carle, G., Annavaram, M.: PIGEON: A high throughput framework for private inference of neural networks using secure multiparty computation. Proc. Priv. Enhancing Technol. (2025). https://doi.org/10.56553/POPETS-2025-0090, https://doi.org/10.56553/popets-2025-0090
13. Hern, A.: Royal free breached uk data law in 1.6m patient deal with google's deepmind. https://www.theguardian.com/technology/2017/jul/03/google-deepmind-16m-patient-royal-free-deal-data-protection-act (2017), the Guardian, Accessed: 2025-09-24
14. Heydari, G., Mahdavi, M., Mala, H.: A review on secure inference from neural networks. SN Comput. Sci. **6**(6), 668 (2025). https://doi.org/10.1007/S42979-025-04190-Z, https://doi.org/10.1007/s42979-025-04190-z
15. Jawalkar, N., Gupta, K., Basu, A., Chandran, N., Gupta, D., Sharma, R.: Orca: Fss-based secure training and inference with gpus. In: IEEE Symposium on Security and Privacy, SP 2024. pp. 597–616. IEEE (2024). https://doi.org/10.1109/SP54263.2024.00063, https://doi.org/10.1109/SP54263.2024.00063
16. Keller, M., Sun, K.: Secure quantized training for deep learning. In: Chaudhuri, K., Jegelka, S., Song, L., Szepesvári, C., Niu, G., Sabato, S. (eds.) International Conference on Machine Learning, ICML 2022. Proceedings of Machine Learning Research, vol. 162, pp. 10912–10938. PMLR (2022), https://proceedings.mlr.press/v162/keller22a.html
17. Koti, N., Pancholi, M., Patra, A., Suresh, A.: SWIFT: super-fast and robust privacy-preserving machine learning. In: Bailey, M.D., Greenstadt, R. (eds.) 30th USENIX Security Symposium 2021. pp. 2651–2668. USENIX Association (2021), https://www.usenix.org/conference/usenixsecurity21/presentation/koti
18. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: A whitepaper (2018), https://arxiv.org/abs/1806.08342
19. Laud, P., Randmets, J.: A domain-specific language for low-level secure multiparty computation protocols. In: 22nd ACM SIGSAC Conference on Computer and Communications Security. CCS '15, Association for Computing Machinery (2015). https://doi.org/10.1145/2810103.2813664, https://doi.org/10.1145/2810103.2813664
20. Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.M., Wang, W.C., Xiao, G., Dang, X., Gan, C., Han, S.: Awq: Activation-aware weight quantization for llm compression and acceleration (2024), https://arxiv.org/abs/2306.00978
21. Lindell, Y.: Secure multiparty computation. Commun. ACM **64**(1), 86–96 (2021). https://doi.org/10.1145/3387108, https://doi.org/10.1145/3387108
22. Lu, T., Zhang, B., Zhang, X., Ren, K.: A new PPML paradigm for quantized models. In: 32nd Annual Network and Distributed System Security Symposium,

NDSS 2025. The Internet Society (2025), https://www.ndss-symposium.org/ndss-paper/a-new-ppml-paradigm-for-quantized-models/
23. Mohassel, P., Rindal, P.: Aby^3: A mixed protocol framework for machine learning. In: Lie, D., Mannan, M., Backes, M., Wang, X. (eds.) PACM SIGSAC Conference on Computer and Communications Security, CCS 2018. pp. 35–52. ACM (2018). https://doi.org/10.1145/3243734.3243760, https://doi.org/10.1145/3243734.3243760
24. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy, SP 2017. pp. 19–38. IEEE Computer Society (2017). https://doi.org/10.1109/SP.2017.12, https://doi.org/10.1109/SP.2017.12
25. Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., van Baalen, M., Blankevoort, T.: A white paper on neural network quantization (2021), https://arxiv.org/abs/2106.08295
26. Ossowski, Y.: Time is running out for eu member states to decide on chat control. https://www.euronews.com/next/2025/09/05/time-is-running-out-for-eu-member-states-to-decide-on-chat-control (2025), euronews, Accessed: 2025-09-24
27. Patra, A., Suresh, A.: BLAZE: blazing fast privacy-preserving machine learning. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020), https://www.ndss-symposium.org/ndss-paper/blaze-blazing-fast-privacy-preserving-machine-learning/
28. Pettai, M., Laud, P.: Automatic Proofs of Privacy of Secure Multi-party Computation Protocols against Active Adversaries. In: Fournet, C., Hicks, M.W., Viganò, L. (eds.) IEEE 28th Computer Security Foundations Symposium, CSF 2015. pp. 75–89. IEEE Computer Society (2015). https://doi.org/10.1109/CSF.2015.13, https://doi.org/10.1109/CSF.2015.13
29. Rathee, D., Bhattacharya, A., Gupta, D., Sharma, R., Song, D.: Secure floating-point training. In: Calandrino, J.A., Troncoso, C. (eds.) 32nd USENIX Security Symposium 2023. pp. 6329–6346. USENIX Association (2023), https://www.usenix.org/conference/usenixsecurity23/presentation/rathee
30. Riazi, M.S., Samragh, M., Chen, H., Laine, K., Lauter, K.E., Koushanfar, F.: XONN: xnor-based oblivious deep neural network inference. In: Heninger, N., Traynor, P. (eds.) 28th USENIX Security Symposium 2019. pp. 1501–1518. USENIX Association (2019), https://www.usenix.org/conference/usenixsecurity19/presentation/riazi
31. Ricciato, F., Kubo, B., Pullonen-Raudvere, P., Kisand, A.D., Talviste, R., Tali, K.: JOCONDE D2.1 Technology Survey and Analysis. Tech. rep., European Union (2025), https://cros.ec.europa.eu/joconde
32. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2015), http://arxiv.org/abs/1409.1556
33. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: Falcon: Honest-majority maliciously secure framework for private deep learning. Proc. Priv. Enhancing Technol. **2021**(1), 188–208 (2021). https://doi.org/10.2478/POPETS-2021-0011, https://doi.org/10.2478/popets-2021-0011
34. Watson, J., Wagh, S., Popa, R.A.: Piranha: A GPU platform for secure computation. In: Butler, K.R.B., Thomas, K. (eds.) 31st USENIX Security Symposium 2022. pp. 827–844. USENIX Association (2022), https://www.usenix.org/conference/usenixsecurity22/presentation/watson

35. Wei, L., Ma, Z., Yang, C., Yao, Q.: Advances in the neural network quantization: A comprehensive review. Applied Sciences **14**(17) (2024). https://doi.org/10.3390/app14177445, https://www.mdpi.com/2076-3417/14/17/7445