Private Neural Network Training with Packed Secret Sharing

Hengcheng Zhou(✉)

Shanghai Jiao Tong University, Shanghai, China
zhc12345@sjtu.edu.cn

Abstract. We present a novel approach for training neural networks that leverages packed Shamir secret sharing scheme. For specific training protocols based on Shamir scheme, we demonstrate how to realize the conversion between packed sharing and Shamir sharing without additional communication overhead. We begin by introducing a method to locally convert between Shamir sharings with secrets stored at different slots. Building upon this conversion, we achieve free conversion from packed sharing to Shamir sharing. We then show how to embed the conversion from Shamir sharing to packed sharing into the truncation used during the training process without incurring additional communication costs. With free conversion between packed sharing and Shamir sharing, we illustrate how to utilize the packed scheme to parallelize certain computational steps involved in neural network training. On this basis, we propose training protocols with information-theoretic security between general n parties under the semi-honest model. The experimental results demonstrate that, compared to previous work in this domain, applying the packed scheme can effectively improve training efficiency. Specifically, when packing 4 secrets into a single sharing, we observe a reduction of more than 20% in communication overhead and an improvement of over 10% in training speed under the WAN setting.

Keywords: Secure multi-party computation · Packed Shamir secret sharing scheme · Neural network training

1 Introduction

Secret-sharing-based Secure Muti-Party Computation (MPC) [20,21] provides a promising solution for privacy-preserving machine learning. With secret sharing, participants first divide their sensitive data into multiple shares. The parties then collaboratively perform the corresponding computation on these shares, with the final results being reconstructed after the computation is completed. Throughout the computation process, data always exist in a shared form, thus protecting data privacy.

Developing generalized neural network training protocols suitable for real-world applications is a challenging yet actively pursued research area in MPC. The initial implementation of private training was carried out by SecureML [15].

SecureML was designed for two parties and was based on the ABY framework [7], which supported the conversion between arithmetic, boolean, and Yao sharing to benefit from different schemes. Following this line of work, several studies implemented privacy-preserving neural network training in scenarios involving three parties [14,18,16,19] and four parties [4,12]. These protocols are tailored for specific numbers of participants, lacking generality and flexibility. We believe that it is crucial to develop protocols that can support an arbitrary number of participants to better accommodate the diverse requirements and scenarios encountered in real-world applications. [2] proposed training protocols between n parties using replicated secret sharing [11]. However, the replicated secret sharing approach suffers from poor scalability. As the number of participants increases, the required communication grows rapidly. [22] introduced neural network training protocols for a general number of n parties based on Shamir secret sharing scheme [17]. While supporting an arbitrary number of parties and offering communication complexity of $\mathcal{O}(n)$, the modulo operations involved in [22] lead to low efficiency and impact its practical usability.

We are interested in training protocols based on Shamir scheme. Compared to additive secret sharing, Shamir scheme offers a threshold feature, enabling a level of error tolerance that is crucial for real-world applications. This tolerance becomes increasingly significant as the number of participants grows, since it becomes challenging to guarantee flawless operation from all involved parties. Replicated secret sharing is another common threshold secret sharing technique, but it suffers from poor scalability. Therefore, leveraging Shamir scheme offers significant benefits. Efficiency is the main factor affecting the practicality of Shamir-based protocols. There has been some work aimed at optimizing protocols based on Shamir sharing to achieve greater efficiency while retaining the flexibility of existing protocols. [1,9] implemented Shamir scheme on Galois rings, which can avoid modulo operations on large prime numbers. Their methods remain at the theoretical level of general computation, and their performance on neural network training is unknown. In this paper, we explore how to accelerate existing protocols based on Shamir sharing by employing the packed Shamir secret sharing scheme [10].

The packed Shamir secret sharing scheme enables packing multiple secrets into a single sharing. After packing, multiple secrets can be computed simultaneously. However, utilizing packed sharing is not straightforward. On one hand, packed secret sharing is well-suited for Single Instruction Multiple Data (SIMD) computations, while the process of neural network training does not entirely align with this assumption, preventing the direct application of packed sharing. On the other hand, there is currently no direct conversion between Shamir sharing and packed sharing. TurboPack [8] demonstrated how to perform general computations on packed sharings, but share conversion in this system requires extra communication.

To address these challenges, we first investigate the training process of neural networks and identify the SIMD portion. Specifically, for each batch, the forward propagation and gradient calculation process for each sample is identical. There-

fore, we can use the packed scheme for these parts and Shamir scheme for the remaining computations. Moreover, we found that share conversion can be implemented without additional communication overhead, with packed-to-Shamir conversion completed locally and Shamir-to-packed conversion embedded into the truncation protocol. Thus, for neural network training protocols based on Shamir sharing, incorporating packed sharing does not require additional communication. Based on these insights, we achieve private training between general n ($n \geqslant 3$) parties $P_1, \cdots, P_n$. The underlying Shamir scheme enables our protocols to support the flexible setting of the number of participants n and adversaries t. To implement multiplication, our protocols need to satisfy $t + s - 1 < n/2$, where s is the number of secrets embedded in a single sharing. Our main contributions are summarized below:

- We propose a sharing conversion framework between Shamir sharing and packed sharing with no additional communication overhead.
- By analyzing the neural network training process, we identify the SIMD portion suitable for the packed scheme. Combined with the free conversion framework, we propose protocols for private neural network training between general n ($n \geqslant 3$) parties using both Shamir sharing and packed sharing.
- We conduct experiments with different numbers of participants and different numbers of secrets to be packed, where we train different neural networks on the MNIST dataset. The results indicate the advantages of our protocols, especially under the WAN setting.

2 Preliminaries

2.1 Packed Shamir Secret Sharing Scheme

The packed Shamir secret sharing scheme is a generalization of the standard Shamir scheme. For a positive integer q, we denote the set $\{1, 2, \cdots, q\}$ by $[q]$. In Shamir scheme, to share a secret $u \in \mathbb{F}_p$ among n participants, the dealer first generates a polynomial $f(x)$ of degree t over $\mathbb{F}_p[x]$ such that $f(0) = u$ and all other coefficients are uniformly random, and then send the share $u_i = f(i)$ to party P_i, $i \in [n]$. The vector $(u_1, \cdots, u_n)$ is called a t-sharing of u, which is denoted by $\langle u \rangle_t$ and $\langle u \rangle$ for simplicity. The secret is stored in evaluation point 0 by default. In our work, we are interested in Shamir sharings with secrets stored at different slots. For a sharing $\langle u \rangle_t$ with corresponding polynomial f, we denote $\langle u \rangle_t$ as $\langle u|_a \rangle_t$ if $f(a) = u$.

For packed Shamir secret sharing, multiple secrets can be shared into one single sharing. Let s be the number of secrets to be packed. A degree-d($d \geqslant s-1$) packed Shamir sharing of $\boldsymbol{x} = (x_1, \ldots, x_s) \in \mathbb{F}_p^s$ is a vector $(w_1, \ldots, w_n)$ for which there exists a polynomial $f(x) \in \mathbb{F}_p[x]$ of degree at most d such that $f(-i+1) = x_i$ for $i \in [s]$, and $f(i) = w_i$ for $i \in [n]$. We denote the vector $(w_1, \ldots, w_n)$ as $[\![\boldsymbol{x}]\!]_d$ and $[\![\boldsymbol{x}]\!]$ for simplicity. We can treat a packed sharing $[\![\boldsymbol{x}]\!]_d$ as a Shamir sharing $\langle x_i|_{-i+1} \rangle_d$ for $i \in [s]$. And we can treat a Shamir sharing $\langle u|_{-i+1} \rangle_d$ as a packed Sharing $[\![\boldsymbol{x}]\!]_d$ satisfying $\boldsymbol{x}_i = u$, $i \in [s]$. For a random

degree-d packed sharing of $\boldsymbol{x}$, any $d-s+1$ shares are independent of $\boldsymbol{x}$. Let $D=t+s-1$, then the sharing $[\![\boldsymbol{x}]\!]_D$ is secure when there are t adversaries. We use $[\![\boldsymbol{A}]\!]_d=(\boldsymbol{W_1},\ldots,\boldsymbol{W_n})\in(\mathbb{F}_p^{m\times r})^n$ to represent the sharing of s matrices with dimension $m\times r$. To avoid ambiguity, we denote $\boldsymbol{A}$ as $\boldsymbol{A}=\left(\boldsymbol{A}_{(1)},\ldots,\boldsymbol{A}_{(s)}\right)\in(\mathbb{F}_p^{m\times r})^s$. Particularly, we call a packed sharing a special sharing, if the s secrets to be packed are the same.

The reconstruction of a degree-d sharing requires $d+1$ shares and can be achieved through the Lagrange interpolation method. For $d\geqslant s-1$ and $\boldsymbol{x},\boldsymbol{y}\in\mathbb{F}_p^s$, $[\![\boldsymbol{x}+\boldsymbol{y}]\!]_d=[\![\boldsymbol{x}]\!]_d+[\![\boldsymbol{y}]\!]_d$, here the addition is coordinate-wise. Let $*$ denote the coordinate-wise multiplication. We can compute $[\![\boldsymbol{x}*\boldsymbol{y}]\!]_{d_1+d_2}=[\![\boldsymbol{x}]\!]_{d_1}*[\![\boldsymbol{y}]\!]_{d_2}$, where d_1 and d_2 need to satisfy $d_1+d_2<n$ so that n participants can recover the secret. Follow the same method in [6] for calculating $\langle ab\rangle_t$ from $\langle a\rangle_t$ and $\langle b\rangle_t$, we can compute $[\![\boldsymbol{x}*\boldsymbol{y}]\!]_{d_3}(d_3\geqslant s-1)$ with the help of a double packed sharing $([\![\boldsymbol{r}]\!]_{d_1+d_2},[\![\boldsymbol{r}]\!]_{d_3})$, where $\boldsymbol{r}\in\mathbb{F}_p^s$ is a random vector. Particularly, a public vector $\boldsymbol{c}\in\mathbb{F}_p^s$ can be transformed to $[\![\boldsymbol{c}]\!]_{s-1}$. Therefore, the parties can compute locally $[\![\boldsymbol{c}+\boldsymbol{x}]\!]_d=[\![\boldsymbol{c}]\!]_{s-1}+[\![\boldsymbol{x}]\!]_d$ and $[\![\boldsymbol{c}*\boldsymbol{x}]\!]_{d+s-1}=[\![\boldsymbol{c}]\!]_{s-1}*[\![\boldsymbol{x}]\!]_d$. For a single element $c\in\mathbb{F}_p$, the parties can compute $[\![c\boldsymbol{x}]\!]_d=c[\![\boldsymbol{x}]\!]_d$.

2.2 Neural Network Training

The training process of a neural network can be summarized into three stages: forward propagation, gradient calculation, and parameter updation. We use a^0 to denote the input. The target parameters that need to be updated are w^l and b^l, where $l\in[L]$ and L is the number of layers. We denote the number of neurons in the l^{th} layer by d^l and use d to denote the number of neurons in the input layer. For fully connected neural networks, forward propagation is performed according to $z^l=w^la^{l-1}+b^l$ and $a^l=\sigma\left(z^l\right)$, where σ is the activation function. We use $\mathcal{L}$ to denote the loss function and define $\delta^l=\partial\mathcal{L}/\partial z^l$, $l\in[L]$. After we compute δ^L, we can compute δ^l according to the formula: $\delta^l=\left(w^{l+1}\right)^T\delta^{l+1}\odot\sigma'\left(z^l\right)$, $l\in[L-1]$. With δ^l, $l\in[L]$, we can compute the gradients $\partial\mathcal{L}/\partial w^l=\delta^l\left(a^{l-1}\right)^T$ and $\partial\mathcal{L}/\partial b^l=\delta^l$. These gradients are then used to update the target parameters w^l and b^l. This three-stage process of forward propagation, gradient calculation, and parameter updation forms the core of the neural network training procedure.

3 Packed Neural Network Training

3.1 Training with Shamir Secret Sharing

We choose protocols from [22] as the underlying Shamir-based protocols and follow the method in them to compute the training of neural networks. In [22], data are represented as fixed-point integers and further encoded in a finite field. Therefore, truncation is required after multiplication. Additionally, truncation is also utilized to implement the calculation of the activation function ReLU. Two kinds of truncation are used in [22]: a probabilistic truncation and a deterministic truncation. The probabilistic approach, which is an efficient method

originally proposed in [3], has been identified by [13] as insecure. Therefore, we use deterministic truncation only to guarantee security.

3.2 Embedding Packed Scheme into the Training Process

We focus on the training of neural networks using the mini-batch gradient descent algorithm. During the training phase, the computation process is identical for each batch. So a natural idea is to bundle different batches together, enabling simultaneous training of multiple batches. While computationally feasible, the results differ from training batches sequentially. For distinct batches, the target parameters requiring updates are different. The target parameters at the beginning of each iteration are the updated parameters from the previous iteration. Therefore, during the neural network training process, the computation of multiple batches is not a parallel process.

In the mini-batch gradient descent algorithm, the average gradients of all samples in a batch are used as an estimate of the global gradients. The gradient calculation procedure is independent for each sample in a batch. And the target parameters requiring updates are the same for each sample. After gradient calculation, the gradients of all samples in a batch are aggregated and averaged to update the parameters. Therefore, we can employ the packed scheme to calculate all processes before gradient aggregation. More precisely, we divide a batch into s sub-batches and pack the s sub-batches together. Here we assume that the batchsize is divisible by s. Through packed sharing, we can simultaneously train s sub-batches. After calculating the gradients, the secrets stored at different slots within the packed sharings must be extracted to complete the gradient aggregation. Once the gradient aggregation is finished, we can get the average gradients to update the parameters.

We refer to the training using solely the Shamir secret sharing scheme as Shamir training and the training utilizing the packed Shamir secret sharing scheme as packed training. The process of an iteration of Shamir training and an iteration of packed training when $s = 4$ is illustrated in Fig. 1. It is important to note that all data are secret-shared during the calculation process. The following theorem shows the equivalence between packed training and Shamir training.

Theorem 1. *Packed training and Shamir training yield identical updates for target parameters after a single iteration.*

Proof. For target parameters $\boldsymbol{w}^l$ and $\boldsymbol{b}^l$, $l \in [L]$, we represent the parameters updated after an iteration of Shamir training as $\boldsymbol{w}^l_{Shamir}$ and $\boldsymbol{b}^l_{Shamir}$, and the parameters updated after an iteration of packed training as $\boldsymbol{w}^l_{packed}$ and $\boldsymbol{b}^l_{packed}$. Let $\boldsymbol{\sigma}^l_i$ denote the gradient corresponding to $\boldsymbol{w}^l$ calculated by the i-th sample in a batch. For Shamir training, $\boldsymbol{w}^l$ is updated as follows:

$$\boldsymbol{w}^l_{Shamir} = \boldsymbol{w}^l - \frac{\eta}{N_B} \sum_{i=1}^{N_B} \boldsymbol{\sigma}^l_i, \tag{1}$$

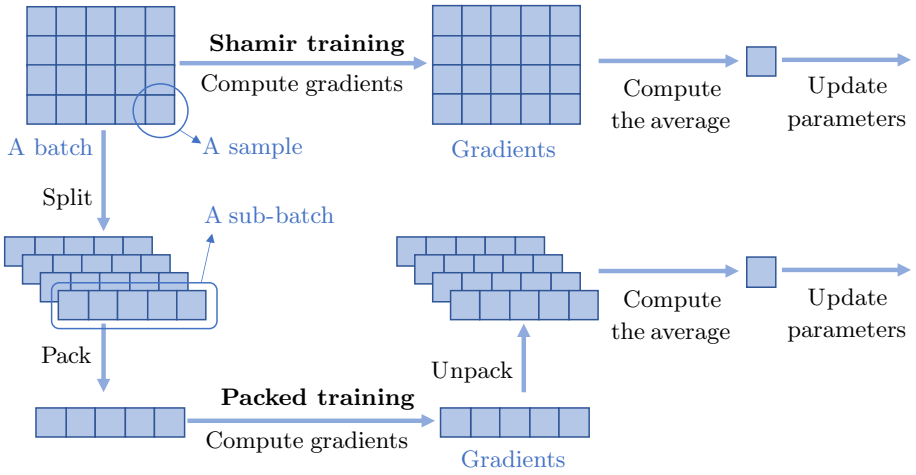

Fig. 1. The process of Shamir training and packed training for a batch ($s = 4$).

where N_B represents the batchsize. For packed training, we assume that N_B is divisible by s and $N_B = sk$. Without loss of generality, when dividing the batch into s groups, we put the j-th sample in the $\lceil j/k \rceil$-th group, $j \in [N_B]$. For $l \in [L]$, the target parameter $\boldsymbol{w}^l$ is updated as follows:

$$\begin{aligned} \boldsymbol{w}^l_{packed} &= \boldsymbol{w}^l - \frac{\eta}{N_B}\Big(\sum_{i=0}^{s-1}\sum_{j=ik+1}^{(i+1)k}\boldsymbol{\sigma}^l_j\Big) \\ &= \boldsymbol{w}^l - \frac{\eta}{N_B}\sum_{i=1}^{N_B}\boldsymbol{\sigma}^l_i \\ &= \boldsymbol{w}^l_{Shamir}. \end{aligned} \tag{2}$$

The same analysis also applies to parameter $\boldsymbol{b}^l$, $l \in [L]$. □

In the process of packed training, we need to convert between Shamir sharing and packed sharing. Two types of conversion are required: (1) After calculating the gradients, we need to aggregate the gradients stored at different slots in packed sharings. Here, we need to first convert the packed sharings to Shamir sharings with secrets stored at the same slot, and then perform corresponding operations on these sharings. (2) Since the target parameters are in the form of packed sharing while the gradients we calculated are in the form of Shamir sharing. When updating the parameters, we need to implement the conversion from Shamir sharing to packed sharing to ensure the consistency of the sharing form. Note that the computations are identical for each sub-batch. Therefore, the target parameters are actually in the form of special sharing, which means we only need the conversion from Shamir sharing to special sharing.

3.3 Free Conversion between Packed Sharing and Shamir Sharing

In this section, we introduce how to implement the two kinds of conversion we need. We begin by introducing a method to locally convert between Shamir sharings with secrets stored at different slots. Building upon this conversion, we achieve a free conversion from packed sharing to Shamir sharing. We then show how to embed the conversion from Shamir sharing to special sharing into the truncation used during the training process without incurring additional communication costs.

Sharing Conversion between Different Slots We first introduce the concept of local conversion between different secret sharing schemes. The following definition comes from [5].

Definition 1 (Share conversion). *Let $\mathcal{S}$, $\mathcal{S}'$ be two secret-sharing schemes over the same secret-domain. We say $\mathcal{S}$ is locally convertible to $\mathcal{S}'$ if there exist local conversion functions $g_1, \ldots, g_n$ such that the following holds. If $(u_1, \ldots, u_n)$ are valid shares of a secret u in $\mathcal{S}$, then $(g_1(s_1), \ldots, g_n(s_n))$ are valid shares of the same secret u in $\mathcal{S}'$. We denote by g the concatenation of all g_i, namely $g(u_1, \ldots, u_n) = (g_1(u_1), \ldots, g_n(u_n))$, and refer to g as a share conversion function.*

Let $\mathcal{S}_\alpha$ denote the Shamir secret sharing scheme with the secret stored at point α. We show that $\mathcal{S}_\alpha$ is locally convertible to $\mathcal{S}_0$.

Theorem 2. *The scheme $\mathcal{S}_\alpha(\alpha \notin [n])$ is locally convertible to $\mathcal{S}_0$ via the function g in which $g_i(u_i) = u_i \prod_{j=1, j\neq i}^{n}(1 - \alpha j^{-1})$, $i \in [n]$.*

Proof. Let $\langle u|_\alpha \rangle_t = (u_1, \cdots, u_n)$ be a t-sharing of $u \in \mathbb{F}_p$, then u can be computed by the Lagrange interpolation method:

$$u = \sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{\alpha - j}{i - j} u_i. \tag{3}$$

Consider the sharing $(g_1(u_1), \ldots, g_n(u_n))$. When applying the Lagrange interpolation method to compute the secret at the evaluation point 0, we can get:

$$\begin{aligned} &\sum_{i=1}^{n} \left(\prod_{j=1, j\neq i}^{n} \frac{-j}{i - j} \right) \left(u_i \prod_{j=1, j\neq i}^{n} \left(1 - \alpha j^{-1}\right) \right) \\ =&\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{-j(1 - \alpha j^{-1})}{i - j} u_i \\ =&\sum_{i=1}^{n} \prod_{j=1, j\neq i}^{n} \frac{\alpha - j}{i - j} u_i \\ =&u, \end{aligned} \tag{4}$$

thus the sharing $(g_1(u_1), \ldots, g_n(u_n))$ is consistent with the same secret u. □

The proof of Theorem 2 only shows that the sharing $(g_1(u_1), \ldots, g_n(u_n))$ is an $(n-1)$-sharing, which means that although we can perform the conversion locally, this comes at the cost of an increased degree of the resulting sharing. It should be noted that since $\alpha \notin [n]$, g is reversible. Therefore, we can get that $\mathcal{S}_0$ is also locally convertible to $\mathcal{S}_\alpha$. Combined with Theorem 2 we can achieve the conversion between Shamir sharings from one slot to another slot. As long as these slots don't coincide with the slots used to deliver the secret share.

From Packed Sharing to Shamir Sharing Since a packed sharing $[\![\boldsymbol{x}]\!]_d$ can be seen as a Shamir sharing $\langle x_i|_{-i+1}\rangle_d$ for $i \in [s]$. We can easily convert the sharing $[\![\boldsymbol{x}]\!]_d$ to s Shamir sharings $\{\langle x_i|_0\rangle_d\}_{i=1}^s$ according to Theorem 2.

From Shamir Sharing to Packed Sharing As discussed in Sect. 3.2, when updating parameters, we need to update s slots in the packed sharings at the same time. This requires the conversion from Shamir sharing to a typical kind of packed sharing, *i.e.*, special sharing. According to Eq. (1), after summing the gradients, we need to perform multiplication with the public value η/N_B. This operation is non-trivial since all data are represented as fixed-point numbers, necessitating a truncation step. We first introduce how to put the comparison used in truncation into the offline phase. Based on this, we introduce how to integrate the requisite sharing conversion within the truncation procedure.

As stated in Sect. 3.1, we use deterministic truncation only. Compared with probabilistic truncation, deterministic truncation eliminates the influence of the carry bit introduced by the addition of random numbers to the bits being truncated [3]. This is achieved through a time-consuming comparison protocol. The input for this comparison consists of a public value and random sharings. Therefore, this process can be performed offline to significantly reduce the online communication overhead of the truncation to a single invocation of the reveal operation. Specifically, for random sharings generated for this truncation, we precompute the comparison results for all potential public values. For a protocol truncating m bits, 2^m comparisons are required. These operations can be parallelized, enabling all comparisons to be completed in constant rounds [3].

The output of deterministic truncation can be seen as the output of probabilistic truncation minus the result of the comparison protocol. And the output of probabilistic truncation is calculated by addition between public values and random sharings [22]. Thus, by expressing the random sharings and the comparison outcome in the form of special sharing, the result of deterministic truncation is automatically in the form of special sharing, all achievable within the offline phase. From the perspective of the online phase, this conversion is free.

Beyond facilitating sharing conversion, deterministic truncation serves the role of degree reduction. This is achieved by adjusting the degree of the random sharings and the comparison output, since these sharings determine the degree of the truncation output. When we convert from packed sharings to Shamir sharings, the degree goes up. However, truncation can allow the degree to return to its original level when we later convert Shamir sharings back to packed sharings.

3.4 Algorithm for Neural Network Training

Utilizing the protocols detailed in [22] along with the sharing conversion process outlined in Sect. 3.3, we are able to perform packed training on neural networks. Algorithm 1 shows the protocol for an iteration of private training using the mini-batch gradient descent algorithm, where η represents the learning rate. We denote this protocol as Π_{DNN}. In Π_{DNN}, $\mathcal{F}_{\delta^L}$ represents the functionality for computing δ^L, and $\mathcal{F}_{DReLU}$ represents the functionality for computing the derivative of ReLU. These functionalities can be realized by the packed version of related protocols documented in [22].

Algorithm 1 Secure Neural Network Packed Training Π_{DNN}

Input: $[\![\boldsymbol{a^0}]\!]_D$, $[\![\boldsymbol{y}]\!]_D$;
for $l = 1$ to $L-1$ **do**
  $[\![\boldsymbol{z}^l]\!]_D = [\![\boldsymbol{w}^l]\!]_D [\![\boldsymbol{a^{l-1}}]\!]_D + [\![\boldsymbol{b}^l]\!]_D$;
  Given $[\![\boldsymbol{z}^l]\!]_D$, the parties invoke $\mathcal{F}_{DReLU}$ to compute $[\![\boldsymbol{e}^l]\!]_D$;
  $[\![\boldsymbol{a}^l]\!]_D = [\![\boldsymbol{z}^l]\!]_D [\![\boldsymbol{e}^l]\!]_D$;
end for
$[\![\boldsymbol{z}^L]\!]_D = [\![\boldsymbol{w}^L]\!]_D [\![\boldsymbol{a}^{L-1}]\!]_D + [\![\boldsymbol{b}^L]\!]_D$;
Given $[\![\boldsymbol{z}^L]\!]_D$ and $[\![\boldsymbol{y}]\!]_D$, the parties invoke $\mathcal{F}_{\delta^L}$ to compute $[\![\boldsymbol{\delta^L}]\!]_D$;
for $l = L-1$ to 1 **do**
  $[\![\boldsymbol{\delta}^l]\!]_D = [\![\left(\boldsymbol{w^{l+1}}\right)^{\boldsymbol{T}}]\!]_D [\![\boldsymbol{\delta^{l+1}}]\!]_D \odot [\![\boldsymbol{e}^l]\!]_D$;
end for
for $l = L$ to 1 **do**
  $[\![\boldsymbol{\partial \mathcal{L} / \partial w^l}]\!]_D = [\![\boldsymbol{\delta^l}]\!]_D [\![\left(\boldsymbol{a^{l-1}}\right)^{\boldsymbol{T}}]\!]_D$;
  The parties convert $[\![\boldsymbol{\partial \mathcal{L} / \partial w^l}]\!]_D$ to $\{\langle \boldsymbol{\partial \mathcal{L} / \partial w^l}_{(j)} | 0 \rangle_{n-1}\}_{j=1}^s$;
  The parties convert $[\![\boldsymbol{\delta}^l]\!]_D$ to $\{\langle \boldsymbol{\delta}^l_{(j)} | 0 \rangle_{n-1}\}_{j=1}^s$;
  $\langle \boldsymbol{\sigma_w}^l \rangle_{n-1} = \sum_{j=1}^s \langle \boldsymbol{\partial \mathcal{L} / \partial w^l}_{(j)} | 0 \rangle_{n-1}$;
  $\langle \boldsymbol{\sigma_b}^l \rangle_{n-1} = \sum_{j=1}^s \langle \boldsymbol{\delta}^l_{(j)} | 0 \rangle_{n-1}$;
  for $i = 1$ to d^l **do**
    $[\![\boldsymbol{b_i}^l]\!]_D = [\![\boldsymbol{b_i}^l]\!]_D - \eta / N_B \sum_{j=1}^{N_B} \langle (\boldsymbol{\sigma_b}^l)_{ij} \rangle_{n-1}$;
  end for
  $[\![\boldsymbol{w}^l]\!]_D = [\![\boldsymbol{w}^l]\!]_D - \eta / N_B \langle \boldsymbol{\sigma_w}^l \rangle_{n-1}$;
end for

When performing local multiplication on packed sharings, the degree of resulting sharings will increase. That's why we need to ensure that $2D = 2(t + s - 1) < n$. After the training is completed, the participants can perform secret reconstruction to obtain the target parameters in plaintext form.

Communication Complexity The training process can be divided into two parts: one that utilizes the packed scheme, such as forward propagation and gradient calculation process for individual samples, and another that relies solely on Shamir secret sharing, such as gradient aggregation and averaging. For the latter, the communication using Shamir training and packed training is the same,

since only Shamir sharings are actually used. It is the first part that causes the communication difference between using Shamir training and packed training. Note that all interactions in our protocols occur within the secret reconstruction operation. During the secret reconstruction process, all participants send their shares to a specific participant, who then recovers the secret and sends it back to the others. For Shamir secret sharing, revealing s secrets requires $2s(n-1)\ell$ bits of messages, where ℓ is the length of p. When considering the packed scheme, to reveal s elements, the parties only need to send one element to the specific party, resulting in a total of $(s+1)(n-1)\ell$ bits of messages with the same communication round as using Shamir secret sharing. Therefore, for the calculation of the first part, as the number of participants increases, the communication overhead of the protocols based on packed scheme tends to be half of that of the protocols based on Shamir scheme.

4 Experiments

The experiments were carried out on a single server equipped with 2 24-core 2.20GHz Intel Xeon Gold 5220R CPUs and 128GB of RAM both in the LAN setting and in the WAN setting. In the WAN setting, the average latency and average bandwidth are set to be 80 ms and 100 MB/s, respectively. We accelerated the computation of local matrix multiplication using an NVIDIA RTX 3090 GPU. We set the prime of the finite field to be $2^{110} - 21$, and the other parameters follow the settings in [22]. We train the same two 3-layer DNNs as in [22], Network A and Network B, on the MNIST dataset. Due to the security issue introduced in [13], we choose protocols in [22] with probabilistic truncation replaced with deterministic truncation and comparison computed offline as the baseline protocols. Our experiments show that the accuracy of Network A can reach 91.49% after 10 epochs of training, and Network B can achieve 92.51% accuracy after 5 epochs of training.

The experiments were first conducted under the LAN setting. We set the number of adversaries t to 1. The time for 12 participants to conduct an epoch of Shamir training using Network A and Network B was $0.98h$ and $0.90h$, respectively. When $s = 4$, the time for one epoch of training using Network A and Network B was $0.99h$ and $0.87h$, respectively. We can see that the training time of Shamir training and packed training are basically the same. This is because training time is influenced by multiple factors, especially the number of communication rounds. However, applying the packed scheme does not change the number of communication rounds. The influence of reduced communication on training time can be better reflected in an environment with higher network latency. Therefore, our subsequent experiments were conducted under the WAN setting. Table 1 shows the time required for an epoch of Shamir training and packed training under different settings of n and s in the WAN setting. This table also includes the theoretical amortized communication overhead for each participant.

Table 1. The time (h) and communication (GB) required for an epoch of training.

(n, s)	Network A		Network B	
	Time	Comm.	Time	Comm.
$(6, 1)$	5.34	3.31	4.23	2.88
$(6, 2)$	5.01	2.78	3.91	2.45
$(12, 1)$	9.64	3.64	8.21	3.16
$(12, 2)$	8.79	3.06	7.36	2.70
$(12, 4)$	8.27	2.76	6.90	2.47

It can be observed from Table 1 that when considering a high-latency network, packed training is more efficient than Shamir training because it requires less data communication. This means that packed training can better adapt to the needs of actual application scenarios.

5 Conclusion

In this paper, we propose private neural network training protocols that leverage the packed Shamir secret sharing scheme. With packed sharing, we achieve parallelization of training data samples in a single batch. Our protocols are based on the Shamir sharing-based protocols proposed in [22] and can fully inherit the advantages of flexible deployment and $\mathcal{O}(n)$ communication complexity. Compared with [22], our protocols have lower communication overhead, making them more suitable for practical scenarios.

Acknowledgments. The work was supported in part by the National Key Research and Development Program of China under Grant 2022YFA1004900, and the National Natural Science Foundation of China under Grants 12031011 and 12361141818.

References

1. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In: Proceedings of the Theory of Cryptography Conference. pp. 471–501 (Dec 2019)
2. Baccarini, A.N., Blanton, M., Yuan, C.: Multi-party replicated secret sharing over a ring with applications to privacy-preserving machine learning. Proceedings on Privacy Enhancing Technologies **2023**, 608–626 (2023)
3. Catrina, O., Hoogh, S.: Improved primitives for secure multi-party integer computation. In: International Conference on Security and Cryptography for Networks. pp. 182–199 (2010)
4. Chaudhari, H., Rachuri, R., Suresh, A.: Trident: efficient 4PC framework for privacy preserving machine learning. In: Symposium on Network and Distributed System Security (NDSS) (2020)

5. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Proceedings of the Second International Conference on Theory of Cryptography. pp. 342–362. Springer-Verlag, Berlin, Heidelberg (2005)
6. Damgård, I., Nielsen., J.B.: Scalable and unconditionally secure multiparty computation. In: Annual International Cryptology Conference. pp. 572–590 (2007)
7. Demmler, D., Schneider, T., Zohner, M.: ABY - a framework for efficient mixed-protocol secure two-party computation. In: Network & Distributed System Security Symposium (2015)
8. Escudero, D., Goyal, V., Polychroniadou, A., Song, Y.: TurboPack: Honest majority mpc with constant online communication. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 951–964 (2022)
9. Escudero, D., Xing, C., Yuan, C.: More efficient dishonest majority secure computation over $\mathbb{Z}_{2^k}$ via galois rings. In: Advances in Cryptology – CRYPTO 2022. pp. 383–412. Springer-Verlag, Berlin, Heidelberg (2022)
10. Franklin, M., Yung, M.: Communication complexity of secure computation (extended abstract). In: Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing. pp. 699–710 (1992)
11. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electronics and Communications in Japan Part III-Fundamental Electronic Science **72**, 56–64 (1989)
12. Koti, N., Patra, A., Rachuri, R., Suresh, A.: Tetrad: Actively secure 4PC for secure training and inference. In: Symposium on Network and Distributed System Security (NDSS) (2022)
13. Li, Y., Duan, Y., Huang, Z., Hong, C., Zhang, C., Song, Y.: Efficient 3PC for binary circuits with application to maliciously-secure DNN inference. In: Proceedings of the 32nd USENIX Conference on Security Symposium (2023)
14. Mohassel, P., Rindal, P.: ABY^3: a mixed protocol framework for machine learning. In: Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. pp. 35–52 (2018)
15. Mohassel, P., Zhang, Y.: SecureML: a system for scalable privacy-preserving machine learning. In: IEEE Symposium on Security and Privacy. pp. 19–38 (2017)
16. Patra, A., Suresh, A.: BLAZE: blazing fast privacy-preserving machine learning. In: Symposium on Network and Distributed System Security (NDSS) (2020)
17. Shamir, A.: How to share a secret. Communications of the ACM **22**(11), 612–613 (1979)
18. Wagh, S., Gupta, D., Chandran, N.: SecureNN: 3-party secure computation for neural network training. Proceedings on Privacy Enhancing Technologies **2019**(3), 26–49 (2019)
19. Wagh, S., Tople, S., Benhamouda, F., Kushilevitz, E., Mittal, P., Rabin, T.: FALCON: honest-majority maliciously secure framework for private deep learning. Proceedings on Privacy Enhancing Technologies **2021**(1), 188–208 (2021)
20. Yao, A.: Protocols for secure computations. In: 23rd Annual Symposium on Foundations of Computer Science. pp. 160–164 (1982)
21. Yao, A.: How to generate and exchange secrets. In: 27th Annual Symposium on Foundations of Computer Science. pp. 162–167 (1986)
22. Zhou, H.: Information-theoretically secure neural network training with flexible deployment. In: Artificial Neural Networks and Machine Learning – ICANN 2023. pp. 324–336. Springer Nature Switzerland, Cham (2023)