SparseBatch: Communication-efficient Federated Learning With Partially Homomorphic Encryption

Chong Wang[1], Jing Wang[2], Zheng Lou[1], Linghai Kong[2], WeiSong Tao[1], and Yun Wang[2*]

[1] State Grid Jiangsu Electric Power Co. LTD, Nanjing, 210024, China
[2] School of computer science and technology, Southeast University, Nanjing, 211189, China
[3] Nanjing Yunchan Information Technology Co.,Nanjing, 211100, China
[*] Corresponding author. E-mail: ywang_cse@seu.edu.cn

Received: Jul. 14, 2024; Accepted: Aug. 24, 2024

Cross-silo federated learning (FL) enables collaborative model training among various organizations (e.g., financial or medical). It operates by aggregating local gradient updates contributed by participating clients, all the while safeguarding the privacy of sensitive data. Industrial FL frameworks employ additively homomorphic encryption (HE) to ensure that local gradient updates are masked during aggregation, guaranteeing no update is revealed. However, this measure has resulted in significant computational and communication overhead. Encryption and decryption operations have occupied the majority of the training time. In addition, the bit length of ciphertext is two orders of magnitude larger than that of plaintext, inflating the data transfer amount. In this paper, we present a new gradient sparsification method, SparseBatch. By designing a new general gradient correction method and using Lion optimizer's gradient quantization method, SparseBatch combines gradient sparsification and quantization. Experimental results show that compared with BatchCrypt, SparseBatch reduces the computation and communication overhead by 5×, and the accuracy reduction is less than 1

Keywords: Homomorphic encryption, Federated Learning, Gradient sparsification, Gradient quantization, Lion optimizer

http://dx.doi.org/10.6180/jase.202508_28(8).0003

1. Introduction

The development of big data is driving the continuous advancement of artificial intelligence. Data serves as the foundation for model training in the field of artificial intelligence. Traditional machine learning mostly employs centralized methods for training machine learning models, which requires consolidating the training data on the server side. However, the issue of isolated data severely limits the scalability of traditional solutions across various scenarios. Although modern organizations possess abundant data, sharing data directly between them is challenging and may also be constrained by issues such as privacy and security [1–3].

Federated learning [4, 5] offers a novel training approach to effectively address the iso-lated data problem. It enables personalized model training without the need for users to upload their data, by exchanging and aggregating model parameters from different users to complete the overall training task. This also circumvents direct privacy breaches. However, federated learning still does not provide complete privacy protection for user information. While exchanging model parameters instead of directly exchanging private data reduces the risk, there still remains a potential for attackers to infer private data.

Various methods have been proposed to prevent clients from leaking updates during aggregation [6–10]. Among these, additively homomorphic encryption (HE), particularly Paillier [11], has gained attention in cross-silo FL due to its strong privacy protection without compromising

learning accuracy. HE enables direct gradient aggregation on ciphertexts, eliminating the need for decryption. It has been widely adopted in cross-silo FL applications [12–16] and can seamlessly integrate into existing FL frameworks, enhancing the parameter server architecture. Before training, clients synchronize an HE key pair. During training, clients encrypt their gradient updates using the public key and upload ciphertexts to the central server. The server aggregates encrypted gradients and distributes the result. Clients decrypt the gradients with their private keys, update local models, ensuring that only encrypted updates are transmitted, preventing information leakage during aggregation.

While Homomorphic Encryption (HE) offers robust privacy assurances for cross-silo Federated Learning (FL), it involves complex cryptographic operations like modular multiplications and exponentiations, which are computationally intensive. Our evaluation of the test platform reveals that over 80% of the training iteration time is consumed by encryption and decryption processes. Moreover, encryption results in significantly larger ciphertexts, leading to a more than 150× increase in data transfer compared to plaintext. The substantial overhead of HE in encryption and communication poses a significant barrier to enabling cross-silo FL. Many FL applications cannot afford such costs.

By reducing the amount of ciphertext that needs to be transmitted, significant reductions in computational and communication overhead can be achieved. Many work have been devoted to studying this aspect, such as gradient sparsification, pruning, knowledge distillation and others. In this paper, we tackle the encryption and communication bottlenecks created by HE with an algorithm combining gradient sparsification and batching, named as SparseBatch. We use the Lion optimizer to achieve low-error gradient quantization, propose a generally applicable gradient sparsification method that meets scalability requirements, and effectively integrate the high-compression Lion gradient quantization with gradient sparsification. Furthermore, we introduce a gradient sparsification mask negotiation mechanism tailored to HE, as well as a batch processing method for 1-bit gradients under homomorphic encryption.

To evaluate our method's efficacy, we incorporated SparseBatch into FATE, a federated learning framework. We use three representative machine learning applications as benchmarks: a three-layer fully connected neural network FCNN trained on the FMNIST dataset, an AlexNet network trained on the CIFAR10 dataset, and an LSTM model trained on the Reuters dataset, covering both computer vision (CV) and natural language processing (NLP) tasks.

The main contributions of this paper are summarized as follows:

1. We have designed a batch sparsification algorithm called SparseBatch to reduce the computational and communication overhead of additively homomorphic encryption. First, we use the Lion optimizer to quantize the gradients to 1 bit, and then perform gradient sparsification.

2. We analyze the requirements of homomorphic encryption algorithms for the transmission mechanism, propose a negotiation-based secondary aggregation gradient sparsification mechanism, and develop a comprehensive batch processing technique for quantized gradients.

3. Experimental results of CV and NLP tasks indicate 5× reduction in computation overhead and communication overhead compared to BatchCrypt. Meanwhile the accuracy reduction is less than 1%.

2. Backgroud research

In this section, we emphasize strict privacy requirements proposed by cross-silo federated learning. We investigate technologies to meet these needs.

2.1. Cross-Silo Federated Learning

Federated learning is a rising machine learning approach that allows numerous participants to collaboratively train a model without centrally sharing raw data. The fundamental concept of this approach is to safeguard data privacy while enabling data distributed across various locations to partake in model training, thereby tackling several privacy and data ownership issues intrinsic to conventional machine learning methods.

In federated learning, two common application scenarios are prevalent: cross-device federated learning and cross-silo federated learning. Cross-device federated learning is fitting for environments hosting numerous mobile or IoT devices, often characterized by limited computational capabilities and unreliable network connections. Conversely, cross-silo federated learning finds greater relevance in situations where a handful of entities have deinstitutions. In such cases, cross-silo federated learning allows these entities to collectively train a model without the necessity of sharing sensitive data.

This paper will concentrate on cross-silo federated learning. Compared to cross-device federated learning, cross-silo federated learning encounters more stringent requirements concerning privacy and performance.

2.2. Privacy Solutions in Federated Learning

Many strategies have been proposed to protect the privacy of clients participating in federated learning. We will briefly examine these solutions and assess their suitability for cross-silo federated learning (FL).

2.2.1. *Differential Privacy (DP)*

Differential Privacy (DP) adds calibrated noise to protect individual privacy in data analysis. In federated learning, DP might expose gradients to the central server during aggregation, compromising privacy. While suitable for some applications, it may not meet the high privacy demands of cross-silo FL.

2.2.2. *Secure Multi-Party Computation (MPC)*

Secure Multi-Party Computation (MPC) allows multiple parties to collaboratively execute a function using their private inputs without revealing individual input. MPC employs cryptographic protocols to ensure each party only sees the computation's output, maintaining input privacy. However, implementing MPC in cross-silo federated learning is challenging. Developers need to design machine learning algorithms and distribute computations among parties while balancing privacy and performance.

2.2.3. *Secure Aggregation*

Secure aggregation merges client updates in federated learning while preserving privacy by encrypting updates before transmission. However, it faces challenges in cross-silo FL. It allows the server to observe aggregated gradients, risking model privacy. Also, clients need to synchronize secret keys and masks, impacting training synchronicity.

2.2.4. *Homomorphic Encryption (HE)*

Homomorphic Encryption (HE) is a notable advancement in privacy-preserving methods, enabling operations like addition to be conducted directly on encrypted data, eliminating the need for decryption. Recent research highlights the effectiveness of additively homomorphic encryption, such as the commonly used Paillier scheme, as a vital tool for ensuring privacy in cross-silo federated learning (FL). In this strategy, each client encrypts its local updates before sending them to the server for aggregation. The aggregated result is then returned to clients for decryption. HE maintains learning accuracy without additional noise during encryption/decryption, and seamlessly integrating into existing learning systems without extensive modifications. However, HE introduces significant computational and communication overhead, requiring careful consideration in practical implementations.

In additively homomorphic encryption schemes like Paillier, both encryption and decryption are computationally intensive. Besides encryption results in large ciphertexts, leading to considerable communication overhead during data transfer. The ciphertext size is determined by the key size. The minimum secure key size for Paillier is often 2048 bits, while a gradient typically occupies only 32 bits. This means that after encryption, the ciphertext size increases by 64×.

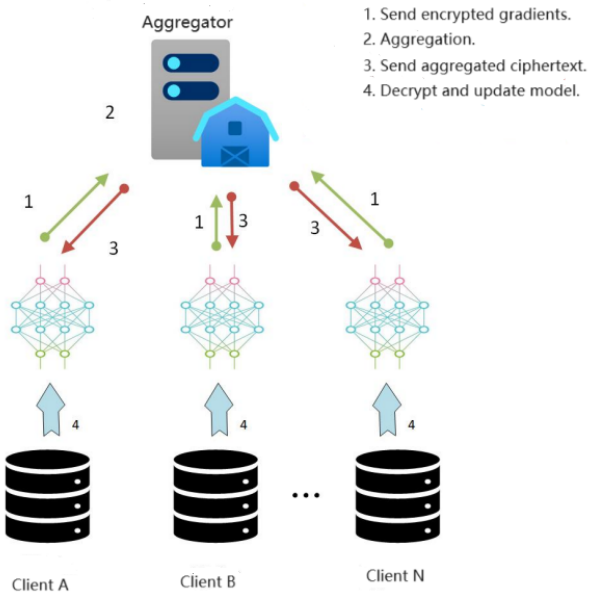

Fig. 1. The architecture of cross-silo FL system using HE.

2.3. Cross-Silo FL Platform with HE

Fig. 1 illustrates a typical cross-silo federated learning system, where Homomorphic Encryption is utilized as a pluggable module on the clients. The aggregator, acting as the server, coordinates clients and aggregates their encrypted gradients. Assuming an honest-but-curious threat model, communications between clients and the aggregator are safeguarded by cryptographic protocols. Before training, the aggregator randomly selects a client leader who generates an HE key pair and syncs it with other clients. The leader also initializes the ML model and distributes weights to all clients. During training, clients compute local gradient updates, encrypt them with the public key, and send results to the aggregator. After receiving updates from all clients, the aggregator aggregates and distributes results to all clients. Clients decrypt aggregated gradients to update their local models.

3. Related work

3.1. BatchCrypt

BatchCrypt [17] is a batching algorithm designed to reduce the computational and communication overhead caused by

additively homomorphic encryption. First, it truncates 32-bit floating-point gradients to the range of $[-\alpha, +\alpha]$. Then, it quantizes each gradient into a r-bit integer. Every batch-size integers are concatenated into a single long integer(a batch). This batch is then encrypted in one go, significantly reducing the number of encryption operations required.

Reviewing the current quantization methods, let's assume we have gradients $g \in [-1, 1]$ that need to be quantized into 8 -bit unsigned integers. The quantization function $\underline{Q(\cdot)}$ and the de-quantization function $Q^{-1}(\cdot)$ are defined as follows:

$$\begin{aligned} Q(g) &= \left\lfloor 255 \cdot \frac{g - \min}{\max - \min} \right\rceil \\ Q^{-1}(q_n) &= q_n \cdot \frac{\max - \min}{255} + n \cdot \min \end{aligned} \tag{1}$$

Let $\lfloor \cdot \rceil$ denotes the standard rounding function. q_n is the sum of n quantized gradients.

To adapt to additively homomorphic encryption, BatchCrypt proposes new requirements for quantization rules:

1. Signed quantization: Gradients are quantized into signed integers, so that positive and negative values cancel each other out, helping to address overflow issues.

2. Quantization interval symmetric about the origin: To ensure that opposite sign numbers can cancel correctly. Otherwise, for example, if $[-1, 1]$ is quantized to $[-128, 127]$, then $(-1 + 1) - > (-128 + 127) = -1$, which results in an error.

3. Uniform quantization. This property makes it suitable for additive homomorphism. In BatchCrypt, the quantization method for gradients $g \in [-\alpha, \alpha]$ is roughly described as mapping $[-\alpha, 0]$ to $[-(2^r - 1), 0]$ and mapping $[0, \alpha]$ to $[0, (2^r - 1)]$. In Fig. 2(a), gradients are quantized into 8-bit integers, while in Fig. 2(b), gradients are quantized into 7-bit integers, utilizing 2 sign bits with two padding bits to distinguish positive and negative overflow. In Fig. 2(b), assuming that each batch concatenates 50 gradients, the total bit number of a batch is 550 bits. Encrypting this batch would reduce the number of encryption operations by $50\times$.

   According to [17], BatchCrypt achieves $23 \times -93\times$ training speedup while reducing the communication overhead by $66 \times -101\times$. The accuracy loss due to quantization errors is less than 1%.

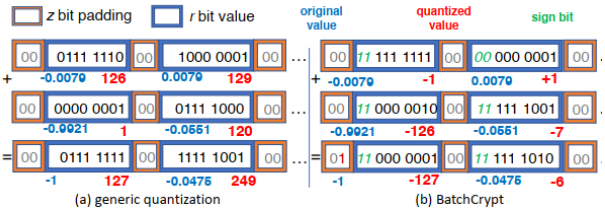

Fig. 2. An illustration of a generic quantization scheme and BatchCrypt.

3.2. Gradient Compression

3.2.1. *Gradient Quantization*

Existing works use gradient compression techniques to reduce network traffic in distributed training [18–20]. Typically, the parameters in models are floating-point numbers. Assuming using 32-bit floating-point numbers, after gradient quantization, parameters can be represented by fewer bits.

Seide et al. proposed 1-bit SGD [21], which quantizes model parameters to 1 bit by only retaining the sign of gradients. To ensure model accuracy, researchers accumulate quantization errors for each batch of data in every training iteration. Experimental results show that such an aggressive quantization strategy, when combined with AdaGrad[22], actually benefits the improvement of model accuracy. Inspired by this, Alistarh et al. proposed QSGD[23]. Unlike previous heuristic algorithms that only quantize gradients exchanged between nodes and do not fully guarantee convergence efficiency, QSGD allows users to dynamically adjust the number of bits sent per iteration by balancing communication bandwidth and convergence time, and it provides convergence guarantees.

When applied to training deep neural networks for tasks such as image classification and automatic speech recognition, QSGD can significantly reduce end-to-end training time. Meanwhile, Wen et al. proposed TernGrad[24], which utilizes ternary gradients to accelerate data parallelism. T TernGrad only requires three values $(-1, 0, 1)$, significantly reducing communication time. By assuming bounded gradients, TernGrad has been mathematically proven to be effective. Experimentally, when TernGrad is applied to AlexNet, it does not incur any precision loss and can even improve precision.

However, even with only 1-bit representation for each parameter, the compression level of gradient quantization is still 10 times, as gradient quantization requires every parameter to participate in transmission.

3.2.2. *Gradient Sparsiflcation*

Unlike gradient quantization, which uses low-bit values to replace floating-point parameters in the original network to achieve compression and acceleration effects, gradient sparsification is similar to pruning techniques in model compression. By selecting parameters through all parameters, only a portion of parameters is transmitted in each parameter synchronization round for acceleration. Unlike directly deactivating parameters after pruning in model compression, gradient sparsification generally does not discard parameters that have not passed the selection. The gradients corresponding to these parameters will accumulate locally and participate in the next round of reselection. Through this accumulation and non-discarding approach, the sparsification level of parameters can be extremely high.

The basic method of gradient sparsity was first proposed by Aji and Heafield[25]. They observed that gradient updates tend to be positively biased and most of them are close to zero. Therefore, they introduced gradient sparsity, keeping 99% of gradient updates local, thereby reducing communication volume by 50 times with the help of coordinate value encoding. However, the compression ratio did not reach the corresponding compression ratio of 99% due to the need for additional bandwidth to represent uploaded and non-uploaded parameter positions. Strom [26] initially used a constant threshold, allowing only gradients larger than a predefined threshold to be uploaded during each training iteration interval.

Through this method, communication volume can be reduced by three orders of magnitude. At this point, the superiority of gradient sparsity over gradient quantization has been demonstrated. To overcome the difficulty of determining a constant threshold, Dryden et al. [27] introduced an adaptive gradient sparsity method using a fixed ratio automatically. Experimental comparisons were made with MPI-Allreduce, and the results showed that the adaptive quantization method proposed in the study performed as well as or better than MPI-Allreduce across multiple models without sacrificing accuracy. It is worth noting that this method achieved close to linear acceleration in data parallel training.

Chia-Yu et al. proposed an adaptive threshold method called AdaComp[28], based on local activity, which involves local selection of residual gradients and automatic adjustment of compression ratios based on local activity. Using this method, fully connected layers can be compressed by 200 times, and convolutional layers by 40 times, while maintaining model accuracy. Lin et al. introduced the DGC[29]method, achieving a compression ratio of 600 times for the entire model, with only 1% of gradients retained. DGC utilizes four methods on the optimizer mSGD: momentum correction, local gradient clipping, momentum factor masking, and preheating strategy, to significantly reduce communication bandwidth without sacrificing accuracy. Unlike simple pruning, Wangni et al. [30] proposed random deletion of model parameters according to coordinates. To compensate for the deleted parameters, the remaining coordinates are appropriately amplified to ensure unbiased gradients.

4. Sparsebatch

This section describes the gradient compression algorithm SparseBatch, which is suitable for federated learning scenarios using semi-homomorphic encryption. Firstly, the Lion[31] optimizer is used to quantize the gradients. Then, a general gradient correction method is designed, along with a gradient sparsification strategy based on a negotiated global mask. The integration of gradient quantization and sparsification has been achieved, resulting in good performance.

4.1. Problem Definition

Consider a Federated Learning (FL) framework comprising a central server and a set of N clients. Each client possesses a specific amount of training data and can communicate with the server. The aim of FL is to acquire a global model without transmitting the training data to the server. This task typically involves solving the subsequent optimization problem in a distributed manner:

$$\min_{x\in\mathbb{R}^d} f(x;\mathcal{D}) \triangleq \frac{1}{n}\sum_{i=1}^{n} f_i\left(x;\mathcal{D}_i\right) \tag{2}$$

where $x \in \mathbb{R}^d$ is the model vector to be learned, $f_i\left(x;\mathcal{D}_i\right) = \frac{1}{n_i}\sum_{j=1}^{n_i} f_{i,j}\left(x;\mathcal{D}_{i,j}\right)$ is the loss function held by the i-th user, $f_{i,j}\left(x;\mathcal{D}_{i,j}\right)$ represents the loss function corresponding to the j-th data sample(or j-th mini-batch dataset) at the i-th client, D_i denotes the dataset stored at the i-th client. We aim to minimize the loss while reducing the communication overhead.

4.2. Design of SparseBatch

4.2.1. *Quantize gradients with Lion optimizer*

Firstly, gradient quantization is implemented from the perspective of the optimizer, using the Lion optimizer. Its update formulas are shown in Eqs. (2) to (5). Here, β_1 and β_2 and are the corresponding optimizer hyperparameters. Equation 4.2 calculates the signbased gradient from the gradients g_t and old momentum m_{t-1} based on the moving average coefficient β_1. Eq. (4) uses this gradient to update the model weights θ. Eq. (5) calculates the new momentum

m_t from the gradients g_t and old momentum m_{t-1} based on the moving average coefficient.

$$u_t = \text{sign}\left(\beta_1 m_{t-1} + (1-\beta_1) g_t\right) \tag{3}$$

$$\theta_t = \theta_{t-1} - \eta_t u_t \tag{4}$$

$$m_t = \beta_2 m_{t-1} + (1-\beta_2) g_t \tag{5}$$

The Lion optimizer uses the sign function to output a sign value instead of a floatingpoint number as the gradient. Using the Lion optimizer can reduce the impact of errors in the gradient quantization process.

4.2.2. General gradient correction method

However, using the Lion optimizer to output quantized gradients presents challenges to the extensibility of existing gradient sparsification methods. Current implementations of gradient sparsification do not support the use of different optimizers. This paper proposes a general gradient correction method.

First, we restate the basic model for an optimizer that updates parameters using a moving average approach:

$$u_t = f\left(n\nabla_t, V_{t-1}\right) \tag{6}$$

$$V_t = g\left(u_t, V_{t-1}\right) = G\left(\eta\nabla_t, V_{t-1}\right) \tag{7}$$

Let ∇ be the original gradient, V be the cached parameter vector used by the optimizer, f be the function representing the process of the optimizer converting the original gradient to the output gradient, and g be the function representing the process of the optimizer updating parameters using the moving average approach. A series of Boolean matrices Mask are used to indicate whether the gradients at corresponding positions pass the threshold filter.

Next, we describe the updates made to various parameters of the model and optimizer when sparsification is directly applied to distributed training. Let r be the local gradient accumulation, and u be the sparsified gradient as a result of the filtering process.

$$\underline{r}_{k,t} = r_{k,t-1} + \eta\nabla_{k,t} \tag{8}$$

$$\underline{r}_{k,t} = r_{k,t-1} + \eta\nabla_{k,t} \tag{9}$$

$$V_{k,t} = G\left(r_{k,t} \cdot \text{ Mask}, V_{k,t-1}\right) \tag{10}$$

$$r_{k,t} = r_{k_k,t} \cdot (\neg \text{ Mask }) \tag{11}$$

This sparsification leads to gradient bias after T iterations of accumulation.

$$u_{k,t+T} = f\left(\eta\left(\nabla_{k,t+T} + \nabla_{k,t+T-1} + \cdots + \nabla_{k,t+1}\right), G'^{(T)}\left(V_{k,t}\right)\right) \tag{12}$$

We use a new correction form to replace the gradient when the sparsification algorithm is not applied.

$$\begin{aligned} &\underline{u}_{k,t+T} = \\ &f\left(\eta\left(\nabla_{k,t+T} + \nabla_{k,t+T-1} + \ldots + \nabla_{k,t+1}\right)\right. \\ &\left.G\left(\eta\left(\nabla_{k,t+T} + \nabla_{k,t+T-1} + \ldots + \nabla_{k,t+1}\right), V_{k_t,t}\right)\right) \end{aligned} \tag{13}$$

This paper approximates this result by changing the position of gradient filtering masks in the original algorithm flow, resulting in the following parameter update process.

$$r_{k,t} = r_{k,t-1} + \eta\nabla_{k,t} \tag{14}$$

$$u_{k,t} = f\left(r_{k,t}, V_{k,t-1}\right) \cdot \text{ Mask} \tag{15}$$

$$V_{k,t} = G\left(r_{k,t}, V_{k,t-1}\right) \cdot \text{ Mask } + V_{k,t-1} \cdot (\neg \text{ Mask }) \tag{16}$$

$$r_{k,t} = r_{k,t} \cdot (\neg \text{ Mask }) \tag{17}$$

In summary, this paper proposes a general gradient sparsification method that overcomes the traditional limitations of gradient sparsification concerning the optimizer. This method can be well integrated with the Lion optimizer. The pseudocode is presented in Algorithm 1

The steps of algorithm 1 are as follows:

Gradient Accumulation

Add the locally computed gradient of this round to the accumulated gradient.

Mask Generation

In line 9, perform threshold filtering on the accumulated gradient values based on equation 4.2, excluding the sign step, to obtain the mask.

Gradient Filtering

In line 10, use masks to filter Lion's output gradients from equation 4.2.

Optimizer Parameter Update

In line 11, apply equation 4.4 to update the momentum. The momentum for unselected gradients remains unchanged.

Reset Accumulated Gradient

Set the accumulated gradient for the selected gradients to zero.

4.2.3. Gradient Sparsiflcation based on negotiated global mask

To enhance privacy protection, the transmission of sparse gradients does not include gradient coordinates. A single gradient aggregation consists of two rounds of communication.

Algorithm 1. Gradient Sparsification with Lion on woker k

Input: Dataset X, initial parameters θ_0, parameters of Lion optimizer $\beta_1,\ \beta_2$;

1: $r_k \leftarrow 0$;
2: $u_k \leftarrow 0$;
3: $m_k \leftarrow \{0, ...\}$;
4: **for** $i = 1, 2, ...$ **do**
5: Sample data x from X;
6: $\nabla_{k,t} \leftarrow \text{Backward}\,(x, \theta_t)$;
7: $r_{k,t} \leftarrow r_{k,t-1} + \eta \nabla_{k,t}$
8: **for** each $m^{(j)}$ **do**
9: $\text{Mask} \leftarrow \text{threshold}\left(\beta_1 m_{k,t-1}^{(j)} + (1-\beta_1)\, r_{k,t}^{(j)}\right)$
10: $u_{k,t}^{(j)} \leftarrow \text{sign}\left(\beta_1 m_{k,t-1}^{(j)} + (1-\beta_1)\, r_{k,t}^{(j)}\right) \odot \text{Mask}$
11: $m_{k,t}^{(j)} \leftarrow \left(\beta_2 m_{k,t-1}^{(j)} + (1-\beta_2)\, r_{k,t}^{(j)}\right) \odot \text{Mask} + m_{k,t-1}^{(j)} \odot \neg\,\text{Mask}$
12: $r_{k,t}^{(j)} \leftarrow r_{k,t}^{(j)} \odot \neg\,\text{Mask}$
13: $u \leftarrow \text{All reduce}\ u_{k,t}$;
14: $\theta_{t+1} \leftarrow SGD\,(\theta_t, u)$;

In the first round, all worker nodes aggregate their masks, negotiating the result by taking the union of each worker node's masks as the global mask. In the second round, the global mask is used to filter the gradients.

Gradients are first grouped, with each group containing m gradients. One group mask corresponds to these m gradients. Using Top-k to filter gradients, if any gradient within these m gradients is selected, the entire group is selected, making the group mask 1 . The server computes the union of all client group masks.

4.2.4. SparseBatch

Now, we summarize the SparseBatch gradient sparsification algorithm based on the Lion optimizer, which is adapted for additive homomorphic encryption. The final pseudocode is shown in Algorithm 2:

Here's the detailed description of Algorithm 2:

Initialization

Each worker node initializes the local model and momentum parameters, with the initial optimizer parameters set to zero. The local sparsity rate is consistent across all worker nodes.

Generate Local Gradients and Masks

At the beginning of each iteration, the worker nodes train the local model using sampled data from their local datasets to obtain the local gradients, which are then added to the accumulated gradient. Subsequently, the accumulated gradient is used as the input gradient, and the baseline gradient is calculated based on the Lion optimizer's formula. In line 10, the mask is computed according to the local sparsity rate.

Mask Aggregation and Sparsity Rate Update

Compress the local masks at a preset ratio, aggregate the masks from all worker nodes, and obtain the union of the masks. In lines 15-19, calculate the actual sparsity rate corresponding to this union, and make slight negative feedback adjustments to the local sparsity rate based on the deviation from the target sparsity rate.

Parameter Update Based on Mask

Use the accumulated gradient as the input gradient, calculate the optimizer's output gradient, and filter it using the mask. The filtered result is the final gradient output by the worker nodes. Calculate the optimizer's parameter updates and filter them using the mask. Apply the parameter updates given by the optimizer to the filtered parts, and retain the original optimizer parameters for the masked parts. Clear the accumulated gradients that passed the mask filter, resetting them to zero.

Gradient Aggregation and Model Update

Each worker node uploads its gradient, obtains the weighted sum result as the actual gradient, and updates their local model accordingly.

5. Evaluation

In this paper, we deploy the SparseBatch prototype on a server in a distributed environment for experimentation. We test the computational and communication overhead of SparseBatch on multiple models and datasets, and compare it with BatchCrypt and plaintext learning to evaluate the extent of overhead optimization achieved by SparseBatch and the impact of gradient compression on model accuracy.

5.1. Setting

The experiment involves 6 clients, distributed across two machines in a data center, each equipped with an 11th Gen Intel Core i7-11700KF CPU. Due to the low overhead of the aggregation operations performed by the server, a standard PC configuration is used for this task. During the experiment, the server connects to the client network via a wide area network (WAN) and is forwarded through a VPN, with actual up and down bandwidths both below 50Mbps. All algorithms are implemented in Python using

Algorithm 2. SparseBatch on worker k

Input: Dataset X, initial parameters θ_0, parameters of Lion optimizer β_1, β_2, target sparsification ratio s', initial worker ratio s_0;

$r_k \leftarrow 0$;
$u_k \leftarrow 0$;
$m_k \leftarrow \{0, ...\}$;
$s \leftarrow s_0$;
for $t = 1, 2, ...$ **do**
  Sample data x from X;
  $\nabla_{k,t} \leftarrow \text{Backward}\,(x, \theta_t)$;
  $r_{k,t} \leftarrow r_{k,t-1} + \eta \nabla_{k,t}$
  for each $m^{(j)}$ **do**
    $\text{Mask}_k^{(j)} \leftarrow \text{threshold}\,[s]\left(\beta_1 m_{k,t-1}^{(j)} + (1-\beta_1)\, r_{k,t}^{(j)}\right)$;
  $Mask'_k \leftarrow \text{compress}\,(Mask)$
  $Mask' \leftarrow All\ reduce\ Mask'_k$
  $Mask \leftarrow uncompress\,(Mask')$
  $s'' \leftarrow \| \text{Mask} \|_0$
  if $s'' < s'$ **then**
    $s \leftarrow dec(s)$
  else
    $s \leftarrow inc(s)$
  for each $m^{(j)}$ **do**
    $u_{k,t}^{(j)} \leftarrow \text{sign}\left(\beta_1 m_{k,t-1}^{(j)} + (1-\beta_1)\, r_{k,t}^{(j)}\right) \odot \text{Mask}$;
    $m_{k,t}^{(j)} \leftarrow \left(\beta_2 m_{k,t-1}^{(j)} + (1-\beta_2)\, r_{k,t}^{(j)}\right) \odot \text{Mask} + m_{k,t-1}^{(j)} \odot \neg\,\text{Mask}$;
    $r_{k,t}^{(j)} \leftarrow r_{k,t}^{(j)} \odot \neg\,\text{Mask}$;
  $u \leftarrow \text{All reduce}\ u_{k,t}$;
  $\theta_{t+1} \leftarrow SGD\,(\theta_t, u)$;

TensorFlow 2.11, with the server's distributed scheduling of clients being implemented via simple RPC calls.

We use 4 representative machine learning applications as benchmarks. A three-layer fully connected neural network (FCNN) is trained on FMNIST, the AlexNet network is trained on CIFAR10, an LSTM model is trained on Reuters, and an ResNet-18 is trained on CIFAR-10.

5.2. Local Performance of SparseBatch

This section evaluates the impact of the proposed gradient sparsification on model accuracy during local model training using the Lion optimizer. For the tested models and datasets, we compare the training results in two scenarios: local training with the Lion optimizer without sparsification, and local training combined with the universal gradient sparsification method. In all experiments, the final sparsity rate is set to 99%, and the FCNN, AlexNet, and LSTM models are trained for 40, 100, and 50 epochs, respectively. The experimental results are displayed using training loss curves and Top-1 accuracy curves.

As shown in Table 1, across all model training experiments, the accuracy achieved after convergence with the sparsified version deviates very little from the non-sparsified version, with differences less than 1%. This confirms the effectiveness of the universal sparsification algorithm.

Fig. 3 shows that training with the sparsified version converges similarly to the nonsparsified version and even demonstrates better convergence speed. For AlexNet and LSTM, the sparsification method even achieves better convergence. This indicates that the combination of quantization and sparsification proposed in this paper not only does not harm convergence but, due to the introduction of additional noise, improves the generalization capability of the model, especially when the model has a large number of parameters.

5.3. Performance Comparison in Federated Learning

In this section, we compare the performance of plaintext learning, BatchCrypt, and Sparse-Batch in federated learning scenario. In all experiments, BatchCrypt's quantization bitwidth is set to 16 bits. SparseBatch is configured with a gradient filtering sparsity rate of 93.75% and a gradient sparsity rate of 95%. Each plaintext in BatchCrypt packages 5016 -bit quantized gradients, while each plaintext in SparseBatch packages 150 items of masks or quantized gradients. The key length for Paillier encryption is 2048 bits. We evaluate the computational and communication overheads, as well as the impact on model accuracy for each method.

Fig. 4 shows the accuracy curves. Experimental results demonstrate that CompressedCrypt has achieved significant improvements over BatchCrypt in both computational and communication overhead across various models. Using the overhead per iteration at training stabilization as a comparison, CompressedCrypt's time expenditure is 11.4%, 5.1%, and 8.6% of the BatchCrypt scheme on FCNN, AlexNet, and LSTM, respectively, while network transmission overhead is only 3.4%, 3.0%, and 3.1%. When comparing the complete training process, CompressedCrypt requires approximately 12.0%, 5.9%, and 9.7% of the time, and 4.8%,3.2%, and 8.4% of the network transmission compared to BatchCrypt for these models. On average, CompressedCrypt's gradient compression achieves a 12 to 20-fold reduction in communication overhead and a significant reduction in computational overhead compared to

Table 1. Comparison of Model Accuracy: Non-Sparsified vs. Sparsified

Model-Method	Accuracy	
FCNN-Lion	89.06%	
FCNN-SparsedLion	88.66%	−0.40%
AlexNet-Lion	79.40%	
AlexNet-SparsedLion	**79.96**%	+0.56%
LSTM-Lion	78.18%	
LSTM-SparsedLion	**78.29**%	+0.11%

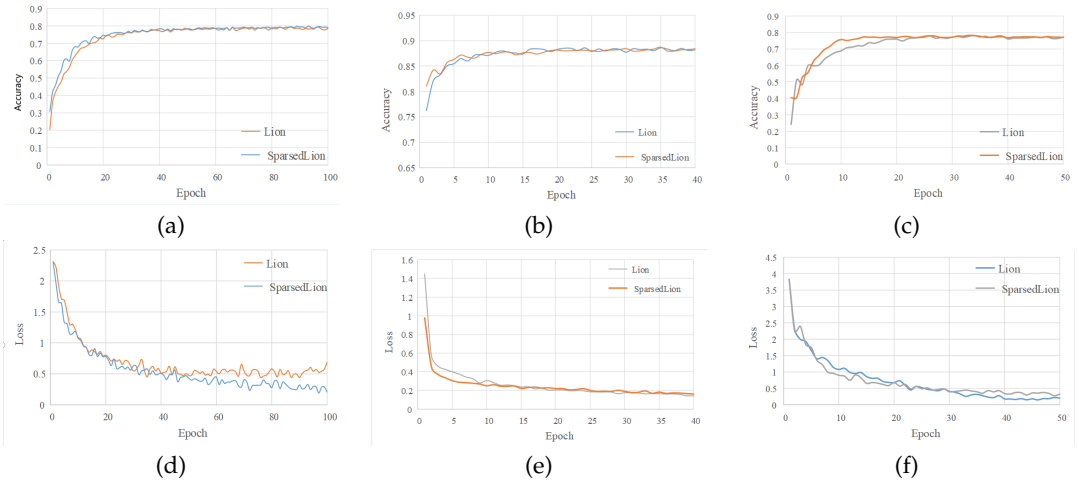

Fig. 3. Performance comparison of Lion and SparsedLion in local training.

BatchCrypt.

Table 2 shows training acceleration effect of SparseBatch across different tasks varies significantly. These differences are primarily related to the complexity of the model and the number of training rounds required. When the model parameters are not extensive, the additional time cost induced by partially homomorphic encryption (PHE) is relatively low, thus resulting in a weaker overall acceleration effect of SparseBatch. Sparse gradients require a warm-up training phase with lower sparsity rates at the beginning of training. If the model converges quickly, the warm-up training phase occupies a larger proportion of the entire process, leading to poorer compression effects. Experiments on ResNet-18 model with a huge number of parameters show the highest acceleration ratio. This comparison supports SparseBatch's better acceleration performance on more complex models, larger datasets, and longer training processes, ensuring the scalability of this method.

6. Conclusion

Horizontal federated learning is a distributed machine learning paradigm, enabling institutions to jointly train models while preserving data privacy. Institutions train models locally and send encrypted parameters to a server, which then aggregates the parameters for model updates. However, the large volume of ciphertext poses challenges in terms of communication overhead. BatchCrypt reduces ciphertext by encoding multiple gradients into a single long integer, while gradient sparsification involves uploading only important gradients. We combines these two approaches and proposes SparseBatch, a communication compression algorithm suitable for additive homomorphic encryption. We use the Lion optimizer to quantize gradients, and design a general gradient correction method along with a gradient sparsification strategy based on a negotiated global mask. Experimental results indicate 5× reduction in computation overhead and communication overhead compared to BatchCrypt with accuracy reduction less than 1%, enhancing the practicality of federated learning protected by partially homomorphic encryption.

Author information

Authors and Affiliations

State Grid Jiangsu Electric Power Co. LTD, Nanjing, 210024, China

Chong Wang & Zheng Lou

School of computer science and technology, Southeast University, Nanjing, 211189, China

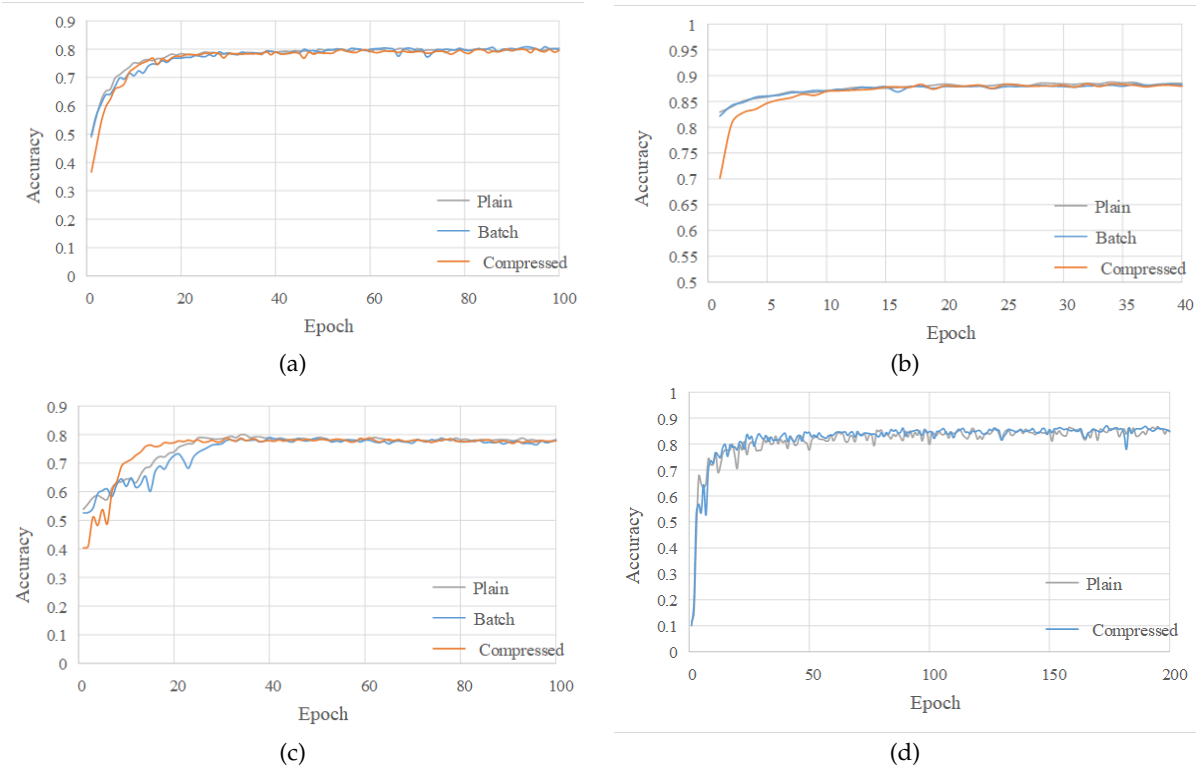

Fig. 4. Performance Comparison of plaintext learning, BatchCrypt and SparseBatch in federated learning.

Table 2. Federated Learning Training Results

Model	Method	Accuracy	Time(s)	Traffic (KB)	Total Time (h)	Total Traffic (GB)
FCNN	plain	88.77%	3.73	769	7.2	4.8
FCNN	batch	88.35%	16.87	1662	32.6	10.5
FCNN	SparseBatch	88.41%	**1.92**	57	3.9	0.5
AlexNet	plain	80.33%	38.09	9774	137.2	121.2
AlexNet	batch	80.83%	131.21	20425	472.7	253.2
AlexNet	SparseBatch	80.19%	6.68	607	**27.9**	8.1
LSTM	plain	**79.98**%	18.36	3958	16.4	8.0
LSTM	batch	78.95%	64.55	8270	57.8	16.6
LSTM	SparseBatch	78.85%	5.53	**245**	5.6	**1.4**
ResNet	plain	86.70%	414.81	87377	2988.3	2167.0
ResNet	batch	86.36%	1534.28	177800	–	–
ResNet	SparseBatch	86.98%	**66.15**	5762	**552.6**	**280.1**

Jing Wang & Linghai Kong & Yun Wang
Nanjing Yunchan Information Technology Co., Nanjing, 211100, China
WeiSong Tao

Contributions

Chong Wang carried out the investigation, methodology, and formal analysis, participated in the Validation, and drafted the manuscript. Jing Wang, Zheng Lou and Linghai Kong participated in the study design and performed the statistical analysis. Yun Wang conceived the study, participated in its design and coordination, and helped draft the manuscript. All authors read and approved the final manuscript.

Ethics declarations

Competing Interests

The author(s) declared no potential conflicts of interest to this article research, authorship, and/or publication.

Ethical and informed consent for data used

Written informed consent for publication of this paper was obtained from the Southeast University and all authors.

Data availability and access

The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

References

[1] P. Bukaty. *The california consumer privacy act (ccpa): An implementation guide*. IT Governance Ltd, 2019.

[2] E. Monitor, (2022) *"Regulation 2022/1925 - Contestable and fair markets in the digital sector and amending Directives (EU) 2019/1937 and (EU) 2020/1828 (Digital Markets Act)"* **Official Journal of the European Union *L***(265): 1–66.

[3] R. Creemers, (2023) *"Cybersecurity Law and regulation in China: Securing the smart state"* **China Law and Society Review *6***(2): 111–145.

[4] M. Ye, X. Fang, B. Du, P. C. Yuen, and D. Tao, (2023) *"Heterogeneous federated learning: State-of-the-art and research challenges"* **ACM Computing Surveys *56***(3): 1–44. DOI: 10.1145/3625558.

[5] J. Wen, Z. Zhang, Y. Lan, Z. Cui, J. Cai, and W. Zhang, (2023) *"A survey on federated learning: challenges and applications"* **International Journal of Machine Learning and Cybernetics *14***(2): 513–535. DOI: 10.1007/s13042-022-01647-y.

[6] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E Ali, B. Guler, and S. Avestimehr, (2022) *"Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning"* **Proceedings of Machine Learning and Systems *4***: 694–720. DOI: 10.48550/arXiv.2109.14236.

[7] J. Kim, S. Kim, J. Choi, J. Park, D. Kim, and J. H. Ahn. "SHARP: A short-word hierarchical accelerator for robust and practical fully homomorphic encryption". In: *Proceedings of the 50th Annual International Symposium on Computer Architecture*. 2023, 1–15. DOI: 10.1145/3579371.3589053.

[8] S. Lee, G. Lee, J. W. Kim, J. Shin, and M.-K. Lee. "HETAL: Efficient privacy-preserving transfer learning with homomorphic encryption". In: *International Conference on Machine Learning*. PMLR. 2023, 19010–19035. DOI: 10.48550/ARXIV.2403.14111.

[9] K. Zhao, J. Hu, H. Shao, and J. Hu, (2023) *"Federated multi-source domain adversarial adaptation framework for machinery fault diagnosis with data privacy"* **Reliability Engineering & System Safety *236***: 109246. DOI: 10.1016/j.ress.2023.109246.

[10] R. Hu, Y. Guo, and Y. Gong, (2023) *"Federated learning with sparsified model perturbation: Improving accuracy under client-level differential privacy"* **IEEE Transactions on Mobile Computing *23***(8): 8242–8255. DOI: 10.1109/TMC.2023.3343288.

[11] P. Paillier. "Public-key cryptosystems based on composite degree residuosity classes". In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1999, 223–238. DOI: 10.1007/3-540-48910-X_16.

[12] Y. Zhang, K. Tian, Y. Lu, F. Liu, C. Li, Z. Gong, Z. Hu, J. Li, and Q. Xu. *"Reparable Threshold Paillier Encryption Scheme for Federated Learning"*. DOI: 10.21203/rs.3.rs-3453596/v1.

[13] S. Mohammadi, S. Sinaei, A. Balador, and F. Flammini. "Optimized Paillier Homomorphic Encryption in Federated Learning for Speech Emotion Recognition". In: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE. 2023, 1021–1022. DOI: 10.1109/COMPSAC57700.2023.00156.

[14] C. Xu, Y. Wang, and J. Wang. "An Improvement Paillier Algorithm Applied to Federated Learning". In: *2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE. 2023, 1445–1451. DOI: 10.1109/icpads60453.2023.00205.

[15] S. KR and J. Judith. *"An Augmented Salp-swarm Optimization Based on Paillier Federated Multi-layer Perceptron (Pf-mlp) and Homomorphic Encryption Standard (Hes) Techniques for Data Security in Cloud Systems"*. 2023. DOI: 10.21203/rs.3.rs-3123750/v1.

[16] P. Yao, H. Wang, C. Zheng, J. Yang, and L. Wang. "Efficient federated learning aggregation protocol using approximate homomorphic encryption". In: *2023 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE. 2023, 1884–1889. DOI: 10.1109/CSCWD57460.2023.10152829.

[17] C. Zhang, S. Li, J. Xia, W. Wang, F. Yan, and Y. Liu. "BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning". In: *2020 USENIX annual technical conference (USENIX ATC 20)*. 2020, 493–506.

[18] Y. Wang and F. Zhu, (2023) *"Distributed dynamic event-triggered control for multi-agent systems with quantization communication"* **IEEE Transactions on Circuits and Systems II: Express Briefs *71***(4): 2054–2058. DOI: 10.1109/TCSII.2023.3329875.

[19] S. Horváth, D. Kovalev, K. Mishchenko, P. Richtárik, and S. Stich, (2023) *"Stochastic distributed learning with gradient quantization and double-variance reduction"* **Optimization Methods and Software** *38*(1): 91–106. DOI: 10.1080/10556788.2022.2117355.

[20] B. Wan, J. Zhao, and C. Wu, (2023) *"Adaptive message quantization and parallelization for distributed full-graph gnn training"* **Proceedings of Machine Learning and Systems** *5*: 203–218. DOI: 10.48550/arXiv.2306.01381.

[21] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu. "1-bit stochastic gradient descent and its application to data-parallel distributed training of speech dnns". In: *Fifteenth annual conference of the international speech communication association*. 2014. DOI: 10 . 21437 / interspeech.2014-274.

[22] J. Duchi, E. Hazan, and Y. Singer, (2011) *"Adaptive subgradient methods for online learning and stochastic optimization."* **Journal of machine learning research** *12*(7): 2121–2159. DOI: 10.1109/TNN.2011.2146788.

[23] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic. "QSGD: Communication-efficient SGD via gradient quantization and encoding". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, 1707–1718. DOI: 10.48550/arXiv.1610.02132.

[24] W. Wen, C. Xu, F. Yan, C. Wu, Y. Wang, Y. Chen, and H. Li. "Terngrad: Ternary gradients to reduce communication in distributed deep learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 2017, 1508–1518. DOI: 10.48550/arXiv.1705.07878.

[25] A. F. Aji and K. Heafield, (2017) *"Sparse communication for distributed gradient descent"* **arXiv preprint arXiv:1704.05021**: DOI: 10.18653/v1/D17-1045.

[26] N. Strom. "Scalable distributed DNN training using commodity GPU cloud computing". In: *Interspeech*. 2015. DOI: 10.21437/Interspeech.2015-354.

[27] N. Dryden, T. Moon, S. A. Jacobs, and B. Van Essen. "Communication quantization for data-parallel training of deep neural networks". In: *2016 2nd Workshop on Machine Learning in HPC Environments (MLHPC)*. IEEE. 2016, 1–8. DOI: 10.1109/MLHPC.2016.004.

[28] C.-Y. Chen, J. Choi, D. Brand, A. Agrawal, W. Zhang, and K. Gopalakrishnan. "Adacomp: Adaptive residual gradient compression for data-parallel distributed training". In: *Proceedings of the AAAI conference on artificial intelligence*. 2018, 2827–2835. DOI: 10.1609/aaai.v32i1.11728.

[29] Y. Lin, S. Han, H. Mao, Y. Wang, and W. J. Dally, (2017) *"Deep gradient compression: Reducing the communication bandwidth for distributed training"* **arXiv preprint arXiv:1712.01887**:

[30] J. Wangni, J. Wang, J. Liu, and T. Zhang. "Gradient sparsification for communication-efficient distributed optimization". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, 1306–1316. DOI: 10.48550/arXiv.1710.09854.

[31] X. Chen, C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le. "Symbolic discovery of optimization algorithms". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 2140. Curran Associates Inc., 2024, 49205–49233. DOI: 10.48550/arXiv.2302.06675.