Efficient Privacy-Aware Federated Learning by Elimination of Downstream Redundancy

Aditya Lohana, Ansh Rupani, Shubham Rai, *Graduate Student Member, IEEE*, and Akash Kumar, *Senior Member, IEEE*

***Abstract*—*Federated Learning* is a distributed machine learning paradigm which advocates training on decentralized data. However, developing a model centrally involves huge communication/ computation overhead, and presents a bottleneck. We propose a method that overcomes this problem while maintaining the privacy of the participants and the classification accuracy. Our method achieves significant speedups compared to existing methods that employ Homomorphic Encryption. Even pessimistically, we achieve a speedup of 4.81x for classification on the ImageNet dataset with an AlexNet architecture, without compromising the privacy of the participants and the accuracy.**

Keywords: federated learning, neural networks, homomorphic encryption.

I. Introduction

Exposing a machine learning model to large amounts of data during training makes it more robust to mis-classifying data samples not encountered during training. However, aggregating the entire training data at a single location requires considerable resources to transport and store the data. Institutions are now becoming aware about the privacy concerns brought about by sharing data with other partners. DL tasks may employ the use of data that is sensitive in nature, such as customer data of various similar organizations [10]. The privacy of such data should be of utmost importance to DL practitioners.

Federated Learning (FL) [8] has emerged as a viable means to bridge this gap between the model and data, while maintaining the privacy of training data. Federated Learning involves training a single neural network over multiple partner nodes via global co-operation without exposing the data at each node. It means that every node maintains a copy of the entire network, and updates the parameters of the model by applying backpropagation using its own data as well as updates from other nodes in the system. This solves the two concerns raised before, as the data resides on the nodes themselves and only model updates in the form of gradients are exchanged.

Unfortunately, just localising the training data to the participants does not alleviate all of the privacy concerns. It is possible for information about the training data to 'leak' using only the publicly-shared gradients [12]. Several techniques have been employed to prevent 'information leakage' from the gradients. A classical approach of obfuscating the gradients is differential-privacy [4], where Gaussian or Laplacian noise is added to the gradients before sharing. The symmetry of the noise prevents the destabilisation of the learning process. Differential-Privacy provides a trade-off between the privacy of the gradients and the convergence of the model and suffers from sub-optimal performance in terms of accuracy. Researchers have attempted to solve the problem of data privacy while guaranteeing convergence by sharing only a small subset of the local gradients [7]. They show that the model converges even if only a fraction (top $k\%$) of the gradients are shared. However, even this scheme is susceptible to attacks. In [11], Phong et. al. show that it is possible to extract information about the training data from a small subset (~3-4%) of the shared gradients. They proposed a new system that guarantees secure aggregation by using additive *homomorphic encryption* (HE). HE allows for computation on ciphertexts without exposing the underlying plaintexts. HE adds considerable communication and computational overheads making it infeasible in practical settings. In this work, we attempt to mitigate these overheads by reducing the redundancy in the aforementioned system. The major contributions of our work are as follows:

- We develop an approach to reduce the communication costs and training time of Federated Learning (FL) using additive HE by eliminating transmission and processing of redundant gradients.
- We show that our approach doesn't compromise model accuracy and maintains system security.
- We present a detailed experimental study to demonstrate the effect of our techniques over different well-accepted benchmark-datasets across various architectures.

II. Preliminaries

A. Downpour Stochastic Gradient Descent (SGD)

The *Parameter-Server* (PS) model [6] of FL involves two classes of nodes:

- **PS:** The PS is the central orchestrator of the learning process in FL. The PS is responsible for maintaining the up-to-date global model, and the aggregation of gradients received from the participants. The model is stored on the PS as a collection of (*key,value*) pairs for constant time (O(1)) access to each weight in every layer.
- **Client:** The clients are the nodes where the actual training of the global model takes place. Each client holds a copy of the global model and calculates gradients for the model using the backpropagation algorithm [1] on its share of the training data. The clients synchronise their local copies with global model at the PS at regular intervals.

The aim of FL is to minimise the total training loss L calculated using the data on each client node

Aditya Lohana, Ansh Rupani, Shubham Rai and Akash Kumar are with the Chair for Processor Design, Center For Advancing Electronics Dresden, Technische Universität Dresden, 01169 Dresden, Germany (e-mail: shubham.rai@tu-dresden.de; akash.kumar@tu-dresden.de).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/MDAT.2021.3063373, IEEE Design and Test

$$L = \sum_{c \in C} \frac{|X_c|}{|X|} L(X_c, y_c, W^t)$$

(X_c, y_c) form the training data on a client c whereas W^t represents the state of the global model at an iteration t of the training process. $L(X_c, y_c)$ is the training loss calculated using the data present on client c. The contribution of the local loss is scaled using the size of the client's share of the dataset.

Each iteration of training consists of the following atomic steps:

- **Fetch:** The clients request for a copy of the global model from the PS, and receive the entire global model (W^t) at iteration t of the training process.
- **Execute:** The global model is applied to the local copy and the gradients are calculated.

$$\nabla_c = \frac{\partial L(X_c, y_c, W^t)}{\partial W^t}$$

  These local gradients are then pushed to the PS.
- **Aggregate:** The PS updates the global model using the gradients received from each client.

$$W^{t'+1} = W^{t'} - \alpha * \nabla_c$$

Specifically in Downpour SGD [5], these steps are performed by each client asynchronously.

B. Paillier – Additive Homomorphic Encryption

Homomorphic encryption is a cryptosystem that aims to allow calculation on encrypted data. Thus, the confidentiality of the data can be maintained while it is being processed by untrusted parties. A homomorphic cryptosystem is an asymmetric encryption scheme that uses a public key to encrypt data and allows the decryption only by parties who possess the private key. Additionally, it allows all parties with a public key to perform a variety of algebraic operations on the data while it is encrypted. The Paillier encryption scheme [2] is a variant of a Partially Homomorphic System, that allows only a subset of arithmetic operations on ciphertexts, namely addition and scalar multiplication, as compared to Fully Homomorphic schemes that allow both addition and subtraction. For more details on Paillier encryption, interested readers are referred to [2].

C. FL with Paillier Encryption

In this work, we evaluate our proposed scheme against the approach suggested in [11]. Phong et al. propose Paillier Encryption for secure aggregation of gradients received from the clients. Each (key, value) pair in the collection of model weights now holds the encrypted weight in the $value$ field. We assume the presence of a central arbitrator, that is responsible for the distribution of the private key to each client, as well as for placing the encrypted weights on all nodes – both the PS and the clients. Out of the three steps of every iteration, only the aggregation step needs to be modified, as follows:

$$\begin{aligned} W_E^{t+1} &= W_E^t \times Enc(-\alpha \nabla_c) \\ &= Enc(W^t) * Enc(-\alpha \nabla_c) \\ &= Enc(W^t - \alpha \nabla_c) \end{aligned} \tag{1}$$

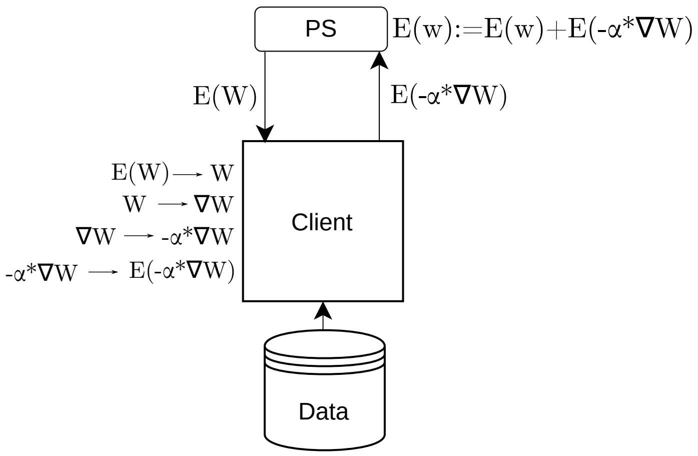


Fig. 1: PS-Client interaction for an iteration in Federated Learning with Additive Homomorphic Encryption

A detailed overview of a single training iteration is described in Fig. 1. Each client first downloads the model and decrypts the weights using its private key. The gradients are calculated, sampled and encrypted before being pushed to the PS. These operations are highlighted on the directed edges between the client and the PS.

D. Threat Model

In this work, we consider a threat model with honest clients and an honest-but-curious PS. This means that the PS is only interested in passively observing the incoming shared gradients and does not attempt to tamper with them. This is consistent with the model assumed in [11]. We also assume the existence of secure channels between the PS and each client node for the transmission of the global model and local gradients.

III. Motivation

The use of additive HE in [11] shifts the accuracy/privacy trade-off in differential privacy to an efficiency/privacy trade-off. As accuracy is an uncompromisable metric in ML tasks, we attempt to mitigate this trade-off by improving the efficiency. Phong et al. incorporated the top-k sparsification method of Shokri and Shmatikov [7] to reduce the ingress traffic at the PS. Due to the sampling of the most significant gradients at the clients, only a small fraction of the gradients are needed to be encrypted and pushed to the PS.

Looking at Fig. 1, we can identify the source of redundancy in this system. The entire model is downloaded by the client during each fetch from the PS. Thus, all the weights in this fetched model must be decrypted, even if δw is zero i.e. the weight hasn't been updated since the last fetch. This 'skew' between the size of gradient-uploads and model-downloads can be exploited to avoid fetching weights that are not important to the training of the model. We achieve this by selectively downloading a subset of the encrypted global model during each local iteration on a client.

IV. Proposed Scheme

Our proposed scheme aims to eliminate redundancy in downstream fetches from PS by transmitting only a subset of the entire model, which is determined by the weights updated between successive fetches by the same client. We maintain dictionaries $LastUpdate$ and $LastFetch$ to keep track of redundant weights, as we need to monitor the following– (i) the last time a weight at the PS was updated by a client; (ii) the last time a client requested the global model. The proposed scheme requires no modification at the client node, as shown in Algorithm 1. However, following changes need to be made to the PS's interaction with each client node:

1) On receiving a sparsified model update from a client, the PS applies these updates to the global model using the rules of Additive Homomorphic Encryption.
2) Apart from aggregation, the PS must also set the $LastUpdate$ entries for each weight in the sparse update to the current global iteration.
3) Whenever the PS receives a fetch-request for the global model from a client, it returns a subset of the weights containing only weights that have been updated since the last fetch request by the same client, i.e. the $LastUpdate$ entry for the weight is greater than the $LastFetch$ entry for the client.
4) Additionaly, the $LastFetch$ for the client is updated to the current global iteration.

The modifications mentioned in points 1 and 2 become a part of the secure aggregation step in Algorithm 2 whereas the other two are incorporated in Algorithm 3. Each iteration consists of the following steps:

1) **Sparse fetch**: The client requests a fetch of the global model from the PS. The PS responds to this request with a sparse model consisting of weights updated since its last fetch (Algorithm 3). At the client node, this sparse update is applied to the local copy of the model to synchronise it with the global model. If this is the first fetch by the client, then it has to download the complete model. This is because each client's $LastFetch$ entry is set to 0 and hence the condition in step 3 of Algorithm 2 ($LastUpdate[w] \geq 0$) is satisfied for all weights w in the model. All subsequent fetches are a subset of the entire global model.
2) **Execute**: The gradients are calculated with respect to the data present on the client using the backpropagation algorithm. The client selects the largest $k\%$ of the gradients, encrypts them and pushes them to the server. (Algorithm 1).
3) **Aggregate and track update**: The gradients received from the client are used to update the global model. The $LastUpdate$ entries for the updated weights are set to the updated global iteration. These weights are available for other clients to fetch as and when requested.

If we consider n clients and a model W, the dictionaries required to track the latest update and fetch histories take $O(|W|)$ and $O(|n|)$ space respectively. The only added overhead in our system arises from filtering the redundant weights in Algorithm 3. We later show in Section V that this overhead at the PS is negligible as compared to the advantage we gain in terms of decrypting fewer weights at each client.

Algorithm 1: Client – Push

```
  Output : Encrypted sparse model $W_E$
1 for $w_E$ in $W_E$ do
2     $w = Dec(w_E)$
3     Update the local model with the decrypted weight:
        $w_C \longleftarrow w$
4 Calculate the gradients $\nabla_C$ using the backpropagation
    algorithm
5 Select the largest $k\%$ of the gradients: $G'$
6 Encrypt the sparse update: $G_E = \{Enc(\alpha * g) | g \in G'\}$
7 Push the encrypted update to the Central Server
```

Algorithm 2: PS – Aggregation

```
  Input         : Encrypted sparse update $G_E$
  Initialization :
  for $w_E$ in $W_E$ do
      Set $LastUpdate[w_E] \longleftarrow 0$
1 for $g_E$ in $G_E$ do
2     Update the global model with the encrypted
        gradient: $w_E^{t+1} \longleftarrow w_E^t - g_E$
3     Set $LastUpdate[w_E] \longleftarrow t + 1$
```

A. Impact on Accuracy

Even though each client only downloads a subset of weights during each fetch, no updated weight is missed during a fetch. Thus, we do not lose any information that may hamper the quality of the trained model. When a full fetch is applied to the local model, it is logically equivalent to applying only the updated global weights because the changes in other weights of the global model are zero. Thus, our scheme does not compromise with the model accuracy achieved in [7, 11]. We provide evidence for our argument in Section V.

B. Security of the Proposed Scheme

The proposed scheme is semantically secure as the Paillier cryptosystem, and hence, is secure and resilient against independent-CPA attacks [3]. Decryption of fetched weights is only possible with a private key, which resides only on clients. Additionaly, the data structures in our scheme ($LastFetch$ and $LastUpdate$) do not reveal any extra information about the gradients or weights. The position of the weights in the model that are updated during the aggregate step are already available to the PS as the model update from the client is a collection of $(key, value)$ pairs. We just leverage this information to sparsify the fetches from the client nodes. Thus, the system is safe against a honest-but-curious PS, but is susceptible to collusion between PS and clients – just like any HE cryptosystem.

V. Experiments

A. Experimental Setup

We validated our scheme by testing it on the following datasets – the MNIST handwritten digits database, the CIFAR-10 dataset and the ImageNet dataset[1]. We conducted experi-

[1]The standard ImageNet dataset (www.image-net.org) contains images from 1000 categories. A random sample of 100 classes was selected to build the dataset.

Algorithm 3: PS - Service Fetch-Request

```
Input :
    • Client-id $c$
    • Global iteration at request by client: $t_g$
    • Parameter update history: $LastUpdate$
    • Client fetch history: $LastFetch$
Output : Encrypted sparse model
Initialization :
for $c$ in Client-pool $C$ do
  | Set $LastFetch[c] \longleftarrow 0$
Initialise the sparse update $W'_E = \{\}$
1 for $w_E$ in $W_E$ do
2   | if $LastUpdate[w_E] \geq LastFetch[c]$ then
3   |   | $W'_E = W'_E \cup \{w_E\}$
4 Set $LastFetch[c] \longleftarrow t_g$
5 Return the sparse encrypted model $W'_E$
```

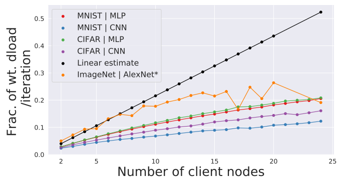
ments for two classes of models – the Multilayer Perceptron and Convolutional Neural Network. All details about the training process such as the network architecture, number of trainable parameters, iterations, learning rate are highlighted in Table 1. Max-pooling was used to down-sample the intermediate feature maps in the CNN. Additionally, we built a subset of AlexNet model[2] (to be referred to as AlexNet* from here on) with 20 million parameters and trained it on a subset of the ImageNet dataset to demonstrate scalability of our approach. All models were trained for 200 global iterations with a decaying learning rate to stabilise training near minima. The experiments were carried out using an extension of the FL framework by Hardy C. [9] with Tensorflow for backpropagation and a Python implementation of the Paillier Encryption scheme. The average time taken for encryption and decryption operations on a 32-bit floating-point number with a key size of 3072 bits is 0.133 seconds and 0.0383 seconds respectively. We can see that the decryption operations are 3 times faster than encryption operations as a random number that is co-prime to the public key needs to be chosen before encrypting each value.

B. Evaluation

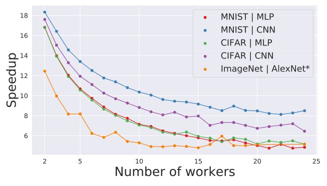
The rationale behind our proposed scheme of sparsifying fetches from the PS was based on the claim made in [7] that certain weights are more important for the training of the model as compared to others. We extended this idea by arguing that a large fraction of the weights remain unaffected during the training process and downloading them during each fetch can be avoided. From Table 1 (Column *Parameters Updated*), it is clear that the majority of the weights are updated not even once in the 200 iterations. This redundancy is especially evident in the CNN architectures with only 18.4%, 16.65% and 15.44% of the models' weights participating in the training process.

Impact of sparse fetches: To investigate the advantage of our scheme, we measure the scalability of our system for different number of client nodes in the system. Federated

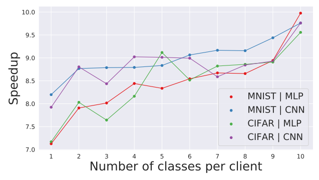
[2]In order to perform classification on 100 output classes, the standard AlexNet model with 80 million parameters was appropriately trimmed to avoid overfitting



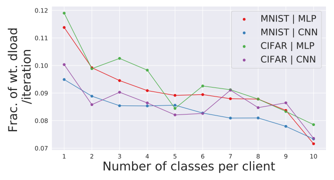
(a)



(b)



(c)



(d)

Fig. 2: (a) Effect of number of clients on relative size of sparse fetches. The growth in size is sub-linear with respect to the number of workers which is a consequence of the overlap of gradients pushed by clients, (b) Effect of number of clients on speedup achieved for training the model for a fixed number of iterations, (c) Effect of data-heterogeneity on speedup achieved for training the model for a fixed number of iterations and clients (d) Effect of number of classes per client on relative size of sparse fetches. The sparse fetches become smaller as the data on each client becomes increasingly homogeneous

Learning leverages parallelism to achieve linear speedup with respect to number of client nodes. We first look at the number of parameters that are downloaded by each client during each fetch as compared to full updates. Fig. 2a demonstrates the effect of our proposed scheme for a range of client nodes in the system. As each client only downloads the weights updated since its last fetch, the expected number of weights downloaded should increase as the number of clients that push gradients to the PS increase. However, as seen in Fig. 2a, the size of the fetched sparse model doesn't grow linearly. This is due to the fact that there exists an overlap in the weights updated by the gradients received from the clients. The upper bound on the relative size of sparse fetches as compared to the full fetch can be deduced from the fraction of weights (Column *Parameters Updated* in Table 1) that are updated throughout training. The upper bound of the relative size of the sparse fetch will be determined by the model architecture and the sparsification factor k.

Impact on training time: Sparsifying the fetches from the PS reduces the number of encrypted weights received by the client. The fetched weights need to be decrypted before they can be applied to the local copy of the model. Thus, reducing the number of weights to be decrypted has a direct effect on training time. Smaller the size of the fetch, faster the decryption. We shift the overhead of decrypting the weights at the client to filtering them at the PS. The decryption step at the client requires $|W_{sparse}|$ decryption operations whereas building a subset of the model requires $|W|$ comparison operators. If the decryption operation takes t_{dec} seconds and comparison takes t_{comp} seconds,

$$T_{sparse} = |W| * t_{comp} + |W_{sparse}| * t_{dec}$$

TABLE 1: An overview of the datasets, models, architecture details, hyperparameters: learning rate and rate of decay, no. of parameters, fraction of weights updated atleast once during NN training, lowest possible training-time speedup and the test accuracy achieved by training models with sparse fetches for 200 iterations respectively. The values in the parentheses represent the number of neurons and the kernel dimensions for the Fully Connected (FC) and Convolutional blocks (Conv) respectively.

Dataset	Model	Architecture	Learn. Rate	Rate of Decay	Parameters	Parameters Updated (%)	Decryp. Time, in Hrs (10 workers)		Total Train. Time, in Hrs (10 workers)		Training Speedup		Acc.
							Sparse dloads	Full dloads [11]	Sparse dloads	Full dloads [11]	10 workers	Lowest obs.	
MNIST	MLP	FC(128)–FC(256)–FC(10)	0.010	0.99	136074	23.04	21	2	21	3	7.00	4.34	0.94
	CNN	C(3x3x32)–P–C(3x3x64)–P–FC(128)–FC(10)	0.010	0.99	421642	18.40	100	6	103	10	10.30	5.43	1.00
CIFAR-10	MLP	FC(128)–FC(256)–FC(10)	0.005	0.95	428938	44.69	70	8	72	10	7.20	2.24	0.88
	CNN	C(3x3x32)–P–C(3x3x64)–P–FC(128)–FC(10)	0.005	0.95	545098	16.85	93	8	96	11	8.72	5.94	0.94
ImageNet*	AlexNet	C(11x11x96)–P–C(5x5x256)–C(3x3x384)–C(3x3x384)–C(3x3x256)-PFC(1024)–FC(100)	0.001	0.99	20627940	15.44	1310	203	1355	248	5.46	4.81	0.86

* C - Convolution layer P - Pooling Layer FC - Fully Connected Layer

We know that $T_{full} = |W| * t_{dec}$. Therefore,

$$\begin{aligned}\frac{T_{sparse}}{T_{full}} &= \frac{|W|*t_{comp} + |W_{sparse}|*t_{dec}}{|W|*t_{dec}} \\ &= \frac{t_{comp}}{t_{dec}} + \frac{|W_{sparse}|}{|W|}\end{aligned} \quad (2)$$

As comparisons are multiple orders of magnitude faster than the decryption operations ($t_{comp} \approx 1.25 * 10^{-7}s$), we can assume $\frac{t_{comp}}{t_{dec}} \approx 0$ with considerable confidence. Additionally, as evident from Table 1 (Column *Parameters Updated*), the sparse fetch will never be bigger than the full model. Thus, a fixed number of iterations with sparse fetches will always be completed faster. Fig. 3 has plots for the time taken (on a logarithmic scale) to run 200 iterations on the global model for varying number of clients which clearly show the advantage of sparse fetches. We can quantify the advantage of sparse fetches over the full fetches by considering the speedup achieved by our scheme ($= \frac{T_{full}}{T_{sparse}}$). As evident from Fig. 2a, $|W_{sparse}|$ (fraction of weights downloaded per iteration) grows as clients in the system increase. Thus, the speedup drops with increasing number of clients. The experimental values of the speedup for a range of workers is depicted in Fig. 2b. As the number of clients in the system increases, the speedup achieved by our system drops from around 17x for 2 clients to 6x for 24 client nodes. We can estimate the lower bound of the speedup achieved by using the fraction of weights that are updated at every iteration. From Equation 2, we can approximate the speedup as the inverse of the relative size of the sparse fetches. These values for the lower bound of the speedup in the case of different models have been populated in Table 1 (Column *Training Speedup-Lowest Obs.*). It is evident that regardless of the number of clients in the system, the speedup achieved by our scheme is at least 2x as compared to [11] for each dataset and model. For the CNN architectures, the speedup is higher (5.43x, 5.94x and 4.81 for the MNIST, CIFAR-10 and ImageNet datasets respectively) due to a lower fraction of weights involved in the training.

Impact of data-heterogeneity: We argued that the sub-linear growth in the size of the sparse fetches observed in Fig. 2a was due to the overlap of the gradients pushed by the clients. Now we attempt to present evidence for this argument and show how the degree of overlap is determined by the distribution of the data among the clients. By the distribution of data, we imply whether the data on each client is *Independent and Identically Distributed* (IID).

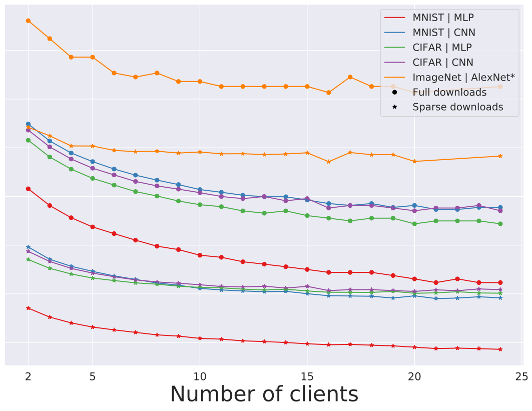
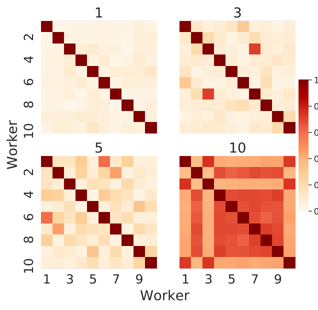
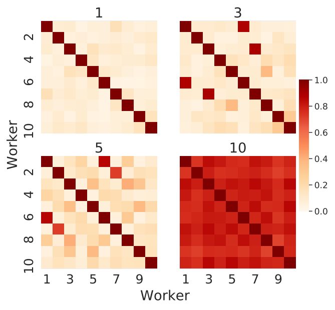


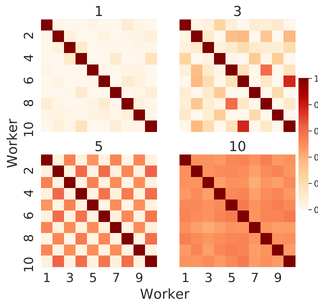
Fig. 3: Time taken for 200 global training iterations for different dataset-model configurations

We will restrict the discussion of data heterogeneity to only the MNIST and the CIFAR-10 datasets for simplicity as both these datasets have 10 output classes. The training data is evenly partitioned into 10 clients. Thus, for an IID setting, each client is randomly assigned a uniform distribution of training data over 10 classes. For non-IID setting, the data is distributed among the clients in such a manner that the data residing on each of them is from a single class. All real-world scenarios lie somewhere between these two settings, and thus, we define the degree of heterogeneity of non-IIDness as the number of classes present on each client (n). This allows us to quantify the heterogeneity of the data distribution when we run our experiments.

The experiments that we conduct are similar to previous section, but with a fixed number of clients (num. of clients = 10). To verify if the relative size of the sparse fetches depends on the heterogeneity of the data, we present the result of the experiments in Fig. 2d. It is clearly evident that the fetches are sparser as the data becomes increasingly homogeneous. The speedup achieved with respect to full fetches for different numbers of classes per client can be inferred from the size of the fetches and is presented in Fig. 2c. The trend matches the relationship that we derive from Equation 2 i.e. the speedup varies inversely with the relative size of the sparse fetches.


(a) MNIST MLP


(b) MNIST CNN


(c) CIFAR-10 MLP

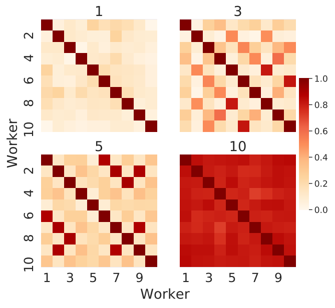

(d) CIFAR-10 CNN

Fig. 4: Jacard Similarity of sparse updates pushed by clients to the PS. The titles of the subplot represent the number of classes of the training data present at each client node.

The gradients pushed to the PS by each client are computed using the data present locally, and hence the gradients pushed by any two clients are similar if the data present on the clients contains samples belonging to common output classes. The heatmaps in Fig. 4 represent the extent of similarity between the gradients pushed by clients. For each heatmap, the cell at (x, y) represents the Jaccard similarity between the gradients pushed by clients x and y. Mathematically,

$$sim(x, y) = \frac{|G_x.keys \cap G_y.keys|}{|G_x.keys \cup G_y.keys|}$$

We can see that as the number of classes per client increase, the gradients become increasingly similar and the fetches become smaller in size. This effect is clear from the cells turning darker as the local datasets become homogeneous.

Model Accuracy: Through our experiments, we have proved that sparsifying the model fetches reduces communications costs and improves training time. As per our argument in Section 5, no important weight is missed while fetching and hence the classification accuracy should theoretically be the same as in [7, 11]. The results of our experiments are presented in Table 1 (Column *Accuracy*). The test accuracy after training each model for 200 iterations gives us excellent results with 94% and 100% classification accuracy for CNN architectures as well as 86% for AlexNet*.

VI. Conclusion and Future Work

Sparsifying downstream updates from the server improves training time and reduces communication cost without degrading the accuracy of the model. The proposed system does not compromise the security of the system in case of a honest-but-curious aggregator (PS). To combat against collusion between PS and edge-nodes, we can combine our proposed approach with other privacy-preserving techniques like differential privacy. Our approach can be incorporated into any scheme that uses sparsified gradient updates. We achieved high speedups, without compromising on the accuracy and privacy.

References

[1] Yann LeCun et al. “A theoretical framework for back-propagation”. In: *Proceedings of the 1988 connectionist models summer school*. Vol. 1. CMU, Pittsburgh, Pa: Morgan Kaufmann. 1988, pp. 21–28.
[2] Pascal Paillier. “Public-Key Cryptosystems Based on Composite Degree Residuosity Classes”. In: *Advances in Cryptology — EUROCRYPT ’99*. Ed. by Jacques Stern. 1999, pp. 223–238.
[3] Dario Catalano, Rosario Gennaro, and Nick Howgrave-Graham. “The Bit Security of Paillier’s Encryption Scheme and Its Applications.” In: *ICTACT*. 2001, pp. 229–243.
[4] Cynthia Dwork. “Differential Privacy: A Survey of Results”. In: *Theory and Applications of Models of Computation*. Ed. by Manindra Agrawal et al. 2008, pp. 1–19.
[5] Feng Niu et al. “HOGWILD! A Lock-Free Approach to Parallelizing Stochastic Gradient Descent”. In: *NIPS*. 2011, 693–701.
[6] Mu Li et al. “Parameter server for distributed machine learning”. In: *Big Learning NIPS Workshop*. Vol. 6. 2013, p. 2.
[7] Reza Shokri and Vitaly Shmatikov. “Privacy-Preserving Deep Learning”. In: *CCS*. 2015, 1310–1321. DOI: 10.1145/2810103.2813687.
[8] Jakub Konečnỳ et al. “Federated learning: Strategies for improving communication efficiency”. In: *arXiv preprint arXiv:1610.05492* (2016).
[9] Hardy C. *AdaComp*. https://github.com/Hardy-c/AdaComp. 2017.
[10] Arthur Jochems et al. “Developing and validating a survival prediction model for NSCLC patients through distributed learning across three countries”. In: *International Journal of Radiation Oncology*Biology*Physics* (2017). DOI: 10.1016/j.ijrobp.2017.04.021.
[11] Le Trieu Phong et al. “Privacy-Preserving Deep Learning via Additively Homomorphic Encryption”. In: *TIFS* (2018), pp. 1333–1345.
[12] Ligeng Zhu, Zhijian Liu, and Song Han. “Deep leakage from gradients”. In: *NIPS*. 2019, pp. 14747–14756.

Aditya Lohana is currently working as a Software Engineer for Microsoft India. He completed his undergraduate studies with a B.E. (2016-2020) in Computer Science from BITS Pilani, Hyderabad. During this time, he spent a semester as Guest Researcher at Technische Universität Dresden with the Chair for Processor Design and worked on privacy-aware distributed Machine Learning. His research interests include Deep Learning, Natural Language Processing and Distributed Systems.

Ansh Rupani received the B.E. degree in Electrical and Electronics Engineering from the Birla Institute of Technology and Science, Pilani, Hyderabad Campus, India in 2018. He is currently working toward the MS degree in Distributed Systems Engineering from Technische Universität, Dresden, Germany. He has worked on emerging reconfigurable technologies and hardware security. He is currently focusing on carrying out distributed inference on heterogeneous edge devices.

Shubham Rai (Graduate Student Member,IEEE) received the BEng degree in electrical and1electronic engineering and MSc degree in physics from the Birla Institute of Technology and Science Pilani, India, in 2011. He is currently working towards the PhD degree with Technische Universität, Dresden, Germany. His research interests include circuit design for reconfigurable nanotechnologies and their logical applications.

Akash Kumar (Senior Member, IEEE)received the joint PhD degree in electrical engineering in embedded systems from the University ofTechnology (TUe), Eindhoven and National University of Singapore (NUS), in 2009. He is currently a professor with Technische Universität Dresden (TUD), Germany, where he is directing the chair for Processor Design. From 2009 to 2015,he was with the National University of Singapore, Singapore. His current research interests include design, analysis, and resource management of low-power and fault-tolerant embedded multiprocessor systems.