Get More for Less in Decentralized Learning Systems

Akash Dhasade*, Anne-Marie Kermarrec*, Rafael Pires*, Rishi Sharma*†, Milos Vujasinovic* and Jeffrey Wigger‡
*EPFL, Switzerland, first.last@epfl.ch
†Corresponding author. ‡Unaffiliated.

***Abstract*—Decentralized learning (DL) systems have been gaining popularity because they avoid raw data sharing by communicating only model parameters, hence preserving data confidentiality. However, the large size of deep neural networks poses a significant challenge for decentralized training, since each node needs to exchange gigabytes of data, overloading the network. In this paper, we address this challenge with JWINS, a communication-efficient and fully decentralized learning system that shares only a subset of parameters through sparsification. JWINS uses wavelet transform to limit the information loss due to sparsification and a randomized communication cut-off that reduces communication usage without damaging the performance of trained models. We demonstrate empirically with 96 DL nodes on non-IID datasets that JWINS can achieve similar accuracies to full-sharing DL while sending up to 64% fewer bytes. Additionally, on low communication budgets, JWINS outperforms the state-of-the-art communication-efficient DL algorithm CHOCO-SGD by up to 4× in terms of network savings and time.**

***Index Terms*—sparsification, compression, communication efficiency, decentralized, P2P, machine learning**

I. INTRODUCTION

Deep learning algorithms have significantly improved artificial intelligence tasks, such as image classification [1], speech recognition [2], and text detection [3]. Enormous volumes of data are produced daily on smartphones and edge devices, which greatly enhance the performance of deep learning models. In addition to facing challenges in efficiently handling the massive size of users' data, the centralized data collection also poses a considerable privacy threat, since personal data can be sensitive to users and its exposure represents a significant concern.

Decentralized learning (DL) has emerged as an attractive alternative for training models on decentralized data, where nodes learn by exchanging models among themselves instead of sharing data with large service providers. This approach provides additional privacy [4]–[7] and obviates the need for complex data management in data centers. Furthermore, DL systems are also popular for their enhanced scalability compared to centralized systems, where the server can become the bottleneck of the system [4]. Nodes in DL are typically devices at the edge of the network, with limited bandwidth, and connected according to a communication topology. While powerful in computational capabilities, these devices must exchange machine learning (ML) models under constrained communication budgets. As a consequence, minimizing the number of bytes transferred when performing DL training has been a major focus of research [6]–[9].

An important technique to minimize the communication during training is *sparsification* or *partial sharing*, which limits the model transfer to a small fraction of *important* parameters [10]–[13]. For sparsification to be effective, it is crucial to select these important parameters with care, ensuring that the resulting global model remains unbiased and maintains the expected accuracy. The rest of the parameters remain local in favor of communication efficiency, thus limiting the information exchanged between nodes. Considering that the data distribution across nodes is often non independent and identically distributed (non-IID), the withheld information due to sparsification can have a detrimental effect on model utility [14]. While there have been in-depth theoretical studies in sparsification-based communication efficiency for DL systems [6], [7], empirical studies with hundreds of nodes and challenging non-IID datasets have been fairly limited. Most settings either use a central coordinator or all-to-all communication [10], [14], [15], which are both unpractical in decentralized systems.

In this paper, we present JWINS (*just what is needed sharing*), a system that performs decentralized training under limited communication budgets while retaining good model performance. The novelty of JWINS lies in the way JWINS addresses the critical issue of choosing important parameters by (1) ranking the parameters and averaging the sparse model vectors in the wavelet-frequency domain, and (2) choosing the size of the sparse model vector using a randomized cut-off scheme which respects the overall communication budget. Since each wavelet coefficient contains information about a subset of the original parameters, JWINS can pack more information in the sparse vectors compared to standard sparsification techniques that rely only on the magnitude of parameter updates. The parameters are both shared and averaged as wavelet coefficients, thus limiting the loss of information due to parameter sparsification in each round. Finally, with the randomized cut-off scheme, each node gets the chance to share the full model parameters every few rounds without violating the communication budget of the entire training process. This shift of approach to the wavelet-frequency domain results in scalable and performant sparsification.

We evaluated JWINS in non-IID data distributions over the extensively used datasets of CIFAR-10, MovieLens, and

the standard LEAF benchmarks of Shakespeare, CelebA and FEMNIST [16]–[18]. Our results for a 96-node training demonstrate that JWINS can achieve an effective sparsification of parameters, reducing the model to 36% of its original size, mostly without impact on test accuracy. In the worst case, JWINS causes an accuracy drop of no more than 3% when compared to full-sharing decentralized learning. In absolute terms, this translates to reductions in network traffic of up to 1.3 TiB against full-sharing. Our scalability study of up to 384 nodes demonstrates that JWINS scales gracefully with increasing number of nodes. To reach the same testing accuracy given a communication budget, JWINS outperforms the sparse communication baseline of random sampling and the state-of-the-art communication compression algorithm called CHOCO-SGD [6] by up to 4× in terms of communication rounds and wall-clock time. Finally, given the exact same communication budget, JWINS reaches 2-10% better accuracies against random sampling sparsification and CHOCO-SGD.

In this paper, we make the following contributions:

- We propose the design of JWINS: a novel decentralized learning algorithm that limits the number of parameters exchanged during the training, building on top of basic sparsification. JWINS relies on a smart combination of wavelet transform and randomized cut-off strategy to limit communication without affecting the accuracy of the learning task.
- We empirically demonstrate that model sparsification and averaging in the wavelet-frequency domain are highly effective in decentralized learning. To the best of our knowledge, we are the first to apply wavelet transform for sparsification and averaging in DL systems.
- We conducted an extensive evaluation study on several learning tasks with non-IID data distributions, including image classification, recommendation, and next character prediction. Our experimental results demonstrate that JWINS outperforms random sampling and CHOCO-SGD, and easily scales up to 384 nodes.

II. BACKGROUND

We briefly describe the DL setup and overview a few communication compression techniques to set the stage for JWINS in Section III.

A. Decentralized learning

a) Objective.: The decentralized learning setting comprises n nodes seeking to jointly learn an ML model. Each node $i \in [n]$ has access to samples from the local data distribution D_i which typically varies across nodes. The goal of learning is to find the parameters of the model $\boldsymbol{x}$ that perform well on the union of datasets spread across nodes by minimizing the average loss, as follows:

$$\min_{\boldsymbol{x}\in\mathbb{R}^d} \left[f(\boldsymbol{x}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{\mathbb{E}_{\xi\sim D_i}[F_i(\boldsymbol{x};\xi)]}_{=:f_i(\boldsymbol{x})} \right] \tag{1}$$

where $f_i(x)$ refers to the local objective function of node i and $F_i(\boldsymbol{x};\xi)$ corresponds to the (possibly non-convex) prediction loss on sample ξ under the model parameters $\boldsymbol{x}$. The local objectives f_i are evaluated as the expected value of loss F_i on the local data distribution D_i. Finally, the nodes in DL are connected through a network topology $G = (V, E)$ which defines the neighborhood of nodes for communication. Every $\{i, j\} \in E$ corresponds to an undirected edge between nodes i and j.

b) Decentralized training.: DL training occurs in repetitions of training rounds. In each round, the nodes follow a general *train-communicate-aggregate* pattern. The training typically consists of running several steps of an optimization algorithm like stochastic gradient descent (SGD) on mini-batches of data sampled from the local data distribution D_i, which is then shared with all (*e.g.*, decentralized parallel stochastic gradient descent (D-PSGD) [4]) or a subset of neighbors (*e.g.*, random model walk (RMW) [19]). Finally, in the aggregation stage, nodes combine the received models with their own, either by performing a plain (RMW) or weighted averaging (D-PSGD). We highlight that JWINS concerns only the *communication* stage in DL, and it is independent of the specific *aggregation* algorithm.

B. Communication compression

Apart from more traditional and general-purpose compression algorithms like LZ4 [20] and LZMA [21], model quantization [22], [23] and sparsification [10], [11], [13] are also referred to as compression techniques in ML. Given that model parameters are floating point numbers, quantization aims at representing such numbers as integers with a smaller bit length. Sparsification, in turn, focuses on reducing the number of parameters or gradients actually shared. JWINS falls into this sparsification category, as we will detail later (Section III).

1) Gradient sparsification.: Having a synchronized model across all worker nodes on every training round, architectures with a central coordinator such as parameter server [24] and federated learning (FL) [25] can afford to share and aggregate gradients rather than parameters. The reason is that worker nodes depart from the same point in the parameter space, and therefore their resulting computed gradients have a common reference point. In such case, sparsification can be trivially done by taking the gradients with the largest magnitude, *i.e.*, all that fall beyond a given threshold. Once the server receives the gradients from workers, it suffices to apply these gradients to the global model before the next training iteration. This method, known as TOPK [12], sets the importance of a parameter by the magnitude of the corresponding gradient in a given round.

a) Sparsification and metadata.: Whenever the gradients for the full model are shared, they can be directly aggregated to the corresponding model parameter according to their position in a serialized vector. Sparsified gradients, on the other hand, do not hold such a positional correspondence to the original model. Hence, one must also send a list of parameter

indexes with which the sparsified gradients are associated. This additional metadata incurs supplementary costs in network usage.

b) Accumulation.: Depending on the model architecture, different parameters may incur changes of distinct magnitudes over the training. For a high degree of sparsification, the gradients corresponding to the same parameters may end up being shared all the time, hence neglecting parts of the model and producing lower-quality models. To avoid this, the gradients $g_i^{(t)}$ of node i at the round t are first accumulated into the residual vector $u_i^{(t)}$ [22], [26], as shown below, where η is the learning rate.

$$u_i^{(t)} = u_i^{(t-1)} - \eta \cdot g_i^{(t)} \tag{2}$$

Subsequently, TOPK is applied on this residual vector, which allows parameters with smaller gradients to be shared at some later point of time. The residual values $u_i^{(t)}$ of shared gradients are reset to zero every time they are selected for sharing.

2) Parameter sparsification.: In contrast to the central-coordinator architectures, there is no central entity for synchronizing models in the decentralized learning setting. Nodes in DL may have distinct models at the beginning of each round, and their computed gradients are incompatible for direct aggregation since these gradients represent the local progress from different points in the parameter space. As a consequence, instead of gradients, in DL the final model parameters must be shared and aggregated. This prevents us from directly applying gradient sparsification techniques in DL.

a) Random sampling.: TOPK sparsification and accumulation from gradient sparsification can be adapted to DL settings for parameter sparsification, but are inefficient [27]. Instead, random sampling is often used [6], [27], [28] where a random subset of the parameters of a predefined size is chosen and shared. Further, when a common pseudo-random generator is used across collaborating nodes, random sampling brings along the benefit of drastically reducing the sparsification metadata. The reason is that just sharing the seed (an integer) from which the random indices are generated suffices for mapping them back to their corresponding model parameters.

III. JWINS

In this section, we describe JWINS and its components in detail. JWINS is built using four core modules. The first one is the representation in the wavelet frequency domain of the model and the model change. The second module is the accumulation of these changes into an importance score for the parameters. These first two modules are outlined in Section III-A, as part of the JWINS parameter ranking. Section III-B then brings the third module by explaining how JWINS selects the best-ranked parameters. The fourth module treats how JWINS compresses the metadata of the sparsified model generated for sharing and is covered in Section III-C. These four core modules are illustrated in Figure 1. Finally, in Section III-D, we present a summary of the JWINS operation,

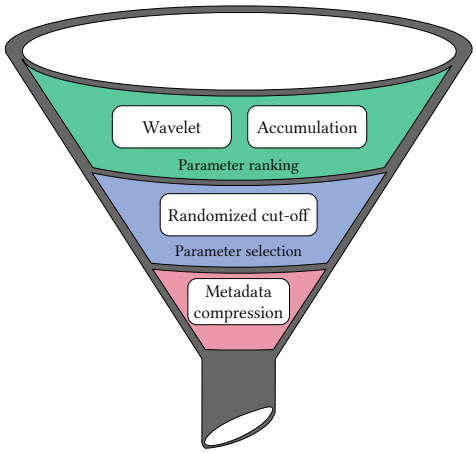

Fig. 1. JWINS consists of four main modules that produce a smaller partial model: *(i)* wavelet transform and *(ii)* accumulation gives importance scores to parameters; *(iii)* randomized cut-off enables nodes to randomly choose the fraction of shared parameters; and *(iv)* metadata compression is used to practically nullify the overheads of metadata when sharing sparsified models.

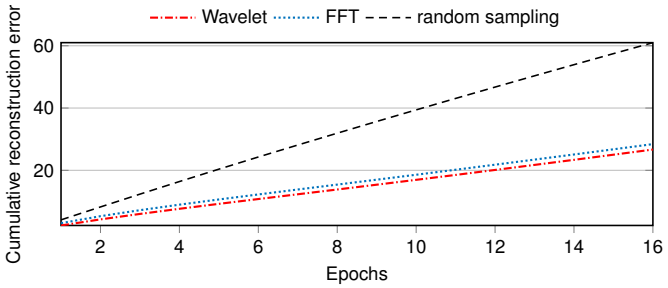

Fig. 2. Mean squared error between the original and reconstructed model when sparsifying parameters using the given algorithms. The plot exhibits the information loss due to sparsification.

detailing how information circulates through the modules and the process of model averaging in the wavelet domain.

In the sections that follow, we use subscripts to index nodes and superscripts to index global rounds numbers and local steps. Therefore $x_i^{(t,s)}$ refers to the model in $t-$th global round after s local steps at node i. Further, in each round $t \in [T]$, nodes perform a total of τ local steps *i.e.*, $s \in [\tau]$.

A. JWINS *parameter ranking*

JWINS parameter ranking generates and maintains a ranking order of the model parameters and relies on the combination of wavelet transform and accumulation. To the best of our knowledge, we are the first to apply them together in decentralized learning systems.

a) Parameter and gradient representation.: Discrete wavelet transform (DWT) and fast Fourier transform (FFT) have been used commonly for image compression [29]–[31]. For instance, JPEG2000 standard [32] uses the properties of DWT for compression. Deriving the inspiration from image compression, we test the performance of compression techniques based on DWT and FFT for neural networks in the presence of sparsification on CIFAR-10 [16] using GN-LeNet [14]

given a communication budget of 10%. Figure 2 presents the cumulative reconstruction error of DWT and FFT against random sampling sparsification for a single node training. This experiment simulates an exchange of model parameters where only a sparsified model is received by applying parameter sparsification in the transformed domain. The reconstruction error is computed as the mean squared error (MSE) of the model without any compression for this round against this sparsified model. Objectively, the algorithm with the lowest cumulative reconstruction error loses the least information on sparsification. Being the compression technique with the least cumulative reconstruction error, we incorporate DWT in JWINS.

In JWINS, we use a four-level discrete wavelet decomposition with Symlet2 (Sym2) wavelets to represent the model $x_i^{(t,s)}$ in the wavelet domain. We experimented with different wavelet functions and Sym2 outperformed the others. Increasing the levels beyond four did not have any noticeable improvements. Intuitively, this representation when compared to the original parameter domain has the advantage of containing coefficients representing the model at lower wavelet frequencies, i.e., single coefficients at higher levels contain information about a neighborhood of parameters in the original domain. Therefore, a sparse wavelet coefficient vector with K non-zero elements packs more information about a change in the model than a vector with K parameters in the original domain.

b) Accumulation in the wavelet domain.: In order to decide which wavelet coefficients should be shared with neighboring nodes, we adapted to the decentralized learning setting the techniques of gradient sparsification and accumulation, as described in Section II-B1. Instead of gradients, JWINS tracks model parameter changes during training and transforms them to the wavelet frequency domain for accumulation. The following equation generates the score:

$$V_i'^{(t)} = V_i^{(t)} + \text{DWT}(x_i^{(t,\tau)} - x_i^{(t,0)}) \tag{3}$$

$V_i'^{(t)}$ represents the new importance score of the model parameters, which is equal to the previously accumulated score $V_i^{(t)}$ added to the difference of the parameters in the wavelet frequency domain over a local training of τ SGD steps. JWINS, however, does not restrict the use of SGD or stateless optimizers. We only use SGD for simplicity. This score from Equation (3) is used by the JWINS parameter selection to share the important parameters. The elements of $V_i^{(t)}$ that correspond to the chosen parameters to be shared are set to zero . The final accumulation step at the end of the round is captured in the equation below:

$$V_i^{(t+1)} = V_i^{(t)} + \text{DWT}(x_i^{(t+1,0)} - x_i^{(t,0)}) \tag{4}$$

where $x_i^{(t+1,0)}$ is the model after the averaging.

B. JWINS *parameter selection*

JWINS parameter selection uses TOPK sparsification on the absolute value of the importance scores generated by JWINS parameter ranking. The size of this selection is governed by a randomized cut-off that determines the subset of model parameters being shared. As noted in Section II-B1, simple TOPK-based sparsification causes the same set of parameters getting repeatedly shared, in detriment of lower-ranked parameters. While this is partly fixed by the parameter ranking based on wavelet and accumulation, JWINS further attenuates this negative effect by sharing random proportions of the model in every round.

To select the number of best-ranked model parameters to be shared, JWINS randomized cut-off component independently chooses in each node a random sharing percentage α from a predetermined distribution based on the available communication budget. Its benefits are threefold: *(i)* parameters that change too slowly can be picked before reaching the threshold; *(ii)* it prevents all nodes from congesting the network by using a large α at the same time; and *(iii)* it prevents a collective drop in model quality due to herd behavior. The negative effect of herd behavior happens when nodes collectively move to a larger α, since they suddenly start sharing parameters that were over-specialized in each individual local data. Averaging these over-specialized parameters together has a similar effect to that of resetting the learning for those parameters, which induces losses in performance. To sum up, several problems arise by having a global sharing fraction α: *(i)* a small α causes the oblivion of slow-changing parameters; *(ii)* large α causes network congestion; and *(iii)* instant global changes in α causes a drop in the quality of the trained model. Put simply, our randomized cut-off strategy avoids them all.

C. Metadata compression

The metadata compression module is a crucial component of JWINS, as it plays a vital role in reducing the amount of data transferred. Without compression and optimization, the metadata from parameter selection would effectively double the number of bytes transmitted, as each parameter is associated with an index. Data compression in DL typically uses floating point compression to perform parameter compression (see Section IV-B for JWINS parameter compression). However, in the case of indices, which are integers, we conducted experiments using various general-purpose compression algorithms. After careful evaluation, we determined that applying Elias gamma [33] compression to the difference array of indices, similar to the approach used in QSGD [23], yielded the best compression rate with a minimal time overhead.

D. JWINS *at work*

We now describe how the various components of JWINS work together. A full pseudocode of JWINS, executed concurrently on each node i is given in Algorithm 1. At the start of round t, the node performs τ training steps on its local data, resulting in the model $x_i^{(t,\tau)}$. The parameter ranking component calculates the model difference, $\Delta x = (x_i^{(t,\tau)} - x_i^{(t,0)})$. This difference is then transformed using DWT and added to an accumulation vector (V) to update rankings for the current round. JWINS parameter selection determines the value of α

Algorithm 1: JWINS on Node i

Input: Initialize $x_i^{(0,0)} = x^{(0,0)}$ and $V_i = \mathbf{0}$; the weight matrix W; metadata compression algorithm C; local dataset D_i; the learning rate η; the number of communication steps T; the set of neighbours N_i; and the number of local SGD steps τ

Output: The trained model $x_i^{(T,0)}$

for $t \leftarrow 0, 1, \dots T$ **do**
  for $s \leftarrow 1$ *to* τ **do**
    Draw a random mini-batch ξ_i from D_i
    $x_i^{(t,s)} \leftarrow x_i^{(t,s-1)} - \eta \cdot \nabla F_i(x_i^{(t,s-1)}; \xi_i)$
  Accumulate $\text{DWT}(x_i^{(t,\tau)} - x_i^{(t,0)})$ in V_i' ▷ eq. (3)
  Randomly choose the communication budget $K_i^{(t)}$
  Get $\text{TOPK}_i^{(t)}$ indices I_i from accumulated V_i'
  Send $\text{DWT}(x_i^{(t,\tau)})[I_i]$ and $C(I_i)$ to all $j \in N_i$
  Receive partial wavelets $Z_j^{(t,\tau)}$ from all $j \in N_i$
  Average all received $Z_j^{(t,\tau)}$ with $\text{DWT}(x_i^{(t,\tau)})$
  Get $x_i^{(t+1,0)}$ by applying DWT^{-1} on the average
  Reset V_i for sent and updated parameters ▷ eq. (4)

for the round based on a randomized cut-off, which determines the percentage of parameters to share. From the accumulation vector, the top $K_i^{(t)}$ indices (I_i) are selected. The corresponding coefficients in $DWT(x_i^{(t,\tau)})$ for these indices, along with $C(I_i)$, the compressed indices, are shared with neighboring nodes in the topology. The received sparse set of wavelet coefficients from all neighbors are averaged using a predefined weighting scheme. The resulting averaged coefficients are then inverted using DWT^{-1} to obtain $x_i^{(t+1,0)}$, *i.e.*, the updated model for the next round. Entries in the accumulation vector (V_i) that were chosen in this round are set to zero. Finally, V_i is updated to account for the model change due to averaging, and the round is concluded.

IV. EVALUATION

We present here an extensive evaluation of JWINS in a decentralized setting. We start by describing JWINS implementation and the experimental setup. Then we present the metrics and results across various experiments. Finally, we discuss the implications of the performance of JWINS.

A. Implementation

JWINS is written in only 800 lines of Python 3.8 code on top of the DecentralizePy framework [34][1]. JWINS uses PyTorch v1.10.2 [35] for ML along with the `torch.multiprocessing` for creating processes. Each process represents a real DL node with its own data and is independent of other processes. ZeroMQ [36] is used to communicate between processes via TCP sockets. The discrete wavelet transformations of the model weights are calculated using the open-source PyWavelets library [37]. We use a common seed for all pseudo-random generators in a node for reproducibility. JWINS is modular, easily extensible, and can support new neural network architectures, datasets, topologies, and compression techniques by plugging other modules in.

[1]Source code is available at https://github.com/sacs-epfl/decentralizepy.

B. Experimental setup

a) Cluster deployment and network: Our experimental setup comprises 6 machines, each equipped with 2 Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz of 8 cores. Hyper-threading is enabled and the machines run Ubuntu 20.04.4 LTS with 5.4.0-99-generic kernel. For each experiment, we run 96 DL nodes (processes) equally distributed across the machines, connected in a random d-regular topology with $d = 4$, *i.e.*, the degree of all vertices is equal to d. For the scalability experiments (Section IV-F), we run up to 384 DL nodes.

b) Learning tasks and hyperparameters: We evaluate JWINS over non-IID train and test datasets of CIFAR-10 [16], MovieLens [17] and the standard FL LEAF benchmarks [18] of CelebA, FEMNIST, and Shakespeare. These datasets are widely used across related works [7], [25], [27], [38]. The image classification task for the first three datasets is achieved over convolutional neural networks of varying specifications [14], [18]. Recommendation over MovieLens is performed through matrix factorization [39] using embeddings. Finally, the next-character prediction for Shakespeare uses a stacked LSTM [18]. This variety of learning tasks and models is chosen to demonstrate the generality of JWINS. The hyperparameters of learning rate (η), number of communication rounds per epoch (r), and batch size (b) are tuned by running a grid search for the full-sharing baseline. This is done separately on data unseen during training. The same values of hyperparameters are then used for JWINS and other baselines. The basic SGD optimizer without momentum is used. The decentralized learning algorithm is D-PSGD over Metropolis-Hastings weights [40].

c) Baselines: We start by comparing JWINS against two decentralized system baselines: *(i)* where the whole set of parameters is exchanged during each communication round (called *full-sharing* in the rest of the paper), and *(ii)* with random sampling sparsification. Among the sparsification algorithms of random sampling and TOPK, we only present random sampling because TOPK overfits to local data as seen in the ablation study (Section IV-E) for JWINS without wavelet (essentially, TOPK). The comparison with full-sharing is meant to assess the performance of JWINS with respect to the resulting accuracy while random sampling is used as a baseline for network savings. Next, we compare JWINS against CHOCO-SGD [6], the state-of-the-art algorithm for communication-efficient decentralized learning with TOPK as the compression technique (referred to as CHOCO hereafter). POWERGOSSIP [9] is another strong communication-efficient algorithm for DL, but it performs as good as tuned CHOCO in their experiments. Hence, we only compare against CHOCO here.

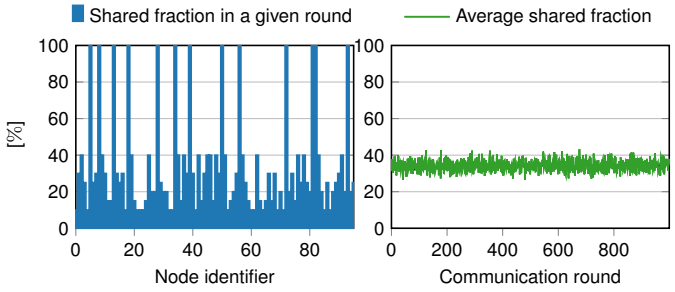

Fig. 3. Randomized cut-off in JWINS. Chart on the left depicts the random percentages selected by JWINS' nodes in a typical communication round. Chart on the right shows the average sharing percentage across nodes over communication rounds.

d) Data partitioning: For the image classification task of CIFAR-10, the dataset is sorted by labels and partitioned into $2n$ shards, where n is the number of nodes. Each node receives 2 shards randomly picked, effectively limiting the maximum number of classes per node to 4, which makes the data distribution non-IID [25]. All other datasets contain a mapping between clients and their data samples. The term *clients* here refers to the users who originally produced a data sample, whereas *nodes* pertains to processes representing a computing unit in our empirical test-bed. For example, the data samples in FEMNIST are grouped by the clients who wrote the digits, and the data samples in MovieLens are grouped by the clients who produced an item-rating pair.

We distribute the data produced by these clients over the 96 training nodes such that each node receives an equal number of clients. Since the Shakespeare dataset contains a very large number of samples, we only distribute 96 out of 660 clients from the original dataset (called Shakespeare onwards), each containing between 800 and 1250 samples. We do this in order to limit the training time and to homogenize the computing requirement per node. This however causes a drop in the reached accuracy, which we consider acceptable for the sake of comparison to full-sharing and random sampling under the same setup.

e) Compression: Due to the large number of model parameters, we empirically assessed multiple compression algorithms in order to reduce the amount of data for transferring these models, and therefore minimize communication. We chose Fpzip compression algorithm [41] since it performed the best across our experiments. This compression is applied uniformly for all the model parameters and for all experiments and baselines. For metadata compression, we use Elias gamma [33] as discussed in Section III-C.

f) Communication: The communication load for full-sharing per iteration is fixed as the full model is always exchanged. In contrast, the communication load for JWINS is randomized but governed by the α distribution used by JWINS parameter selection. In JWINS, $\alpha \in \{10\%, 15\%, 20\%, 25\%, 30\%, 40\%, 100\%\}$, *i.e.*, α is uniformly randomly picked from this list every round by every node independently. Empirically, this works well for all the tasks with minimal accuracy trade-offs. Figure 3 shows this distribution in action in a JWINS run. For a fair comparison of the learning performance of JWINS and random sampling, we limit the communication budget per communication round of random sampling to match the total communication by JWINS. This results in a sharing of 37% of model parameters in every iteration of random sampling. We also evaluate JWINS in settings with lower communication budgets of 20% and 10% against CHOCO.

g) Metrics: We evaluate JWINS along the following metrics: (1) average top-1 test accuracy of nodes on the test set, (2) average loss of nodes on the test set, (3) average number of bytes transferred by nodes. These are real values measured by instrumenting the experiments. Every experiment is conducted five times, using different random seeds that determine the data distribution, neighbors, and initial weights. The values we provide are averages from these five runs, within a 95% confidence interval.

C. JWINS *Performance*

In these experiments, we run full-sharing, random sampling, CHOCO and JWINS for a fixed number of epochs. The number of epochs for the datasets is chosen based on previous relevant work [14], [18], [38]. Since we use a subsampled dataset for Shakespeare, the number of epochs is chosen to match the point where the test loss of full-sharing starts to diverge due to the models of the nodes overfitting. From a systems perspective, communication rounds hold more significance than epochs. Hence, for the accuracy and loss curves, we use communication rounds on the x-axis.

1) Learning accuracy: Figure 4 and Table I show that JWINS is able to achieve as good test accuracy and test loss as full-sharing for most datasets. The performance is slightly off for the CIFAR-10 dataset, but only by 3% in test accuracy. However, JWINS beats random sampling by 2% to 15% in test accuracy. In addition to reaching a better accuracy, JWINS consistently converges faster than random sampling. The fact that JWINS sometimes also outperforms full-sharing is due to the stochasticity of the learning process.

2) Network savings: Our quantifications in Table I for Figure 4 show that JWINS saves over 60% of network usage across all datasets when compared to full-sharing. For CIFAR-10, the difference in accuracy is largely compensated by the 397 GiB of network savings. Thanks to the metadata compression, the amount of metadata in JWINS is negligible compared to the model, depicted in row-3 of Figure 4.

The difference in performances across different datasets can be explained by the underlying data distributions on nodes. For the CelebA and FEMNIST datasets, the nodes likely carry samples of each class although disproportionately under the non-IID distribution. The data partitioning in CIFAR-10 is rather extreme where nodes carry samples from only a few classes (at most 4), making it much harder to learn. Further, the diverging test loss of JWINS for Shakespeare dataset is the typical over-fitting scenario where nodes begin to over-learn on their local datasets while hurting the global test set

TABLE I

FINAL TEST ACCURACIES AND NETWORK TRANSFERS FOR JWINS IN COMPARISON TO FULL-SHARING AND RANDOM SAMPLING FOR PLOTS IN FIGURE 4. JWINS ACHIEVES AS GOOD TEST ACCURACY AS FULL-SHARING ACROSS MOST DATASETS, WHILE IT IS 2%-15% BETTER THAN RANDOM SAMPLING. JWINS SAVES SEVERAL HUNDREDS OF GIGABYTES IN NETWORK TRANSFERS.

DATASET	EPOCHS	TEST ACCURACIES OF ALGORITHMS ↑			TOTAL DATA SENT [GiB] ↓		NETWORK SAVINGS
		full-sharing	random sampling	JWINS	full-sharing	JWINS	
CIFAR-10	268	58.3	40.1	55.3	628.2	231.2	62.2%
MovieLens	400	91.7	89.1	92.6	1103.5	394.6	64.2%
Shakespeare	57	35.0	30.5	34.5	2127.2	753.7	64.6%
CelebA	26	89.7	89.0	90.9	10.4	3.8	63.5%
FEMNIST	25	80.6	79.6	81.6	557.5	199.2	64.3%

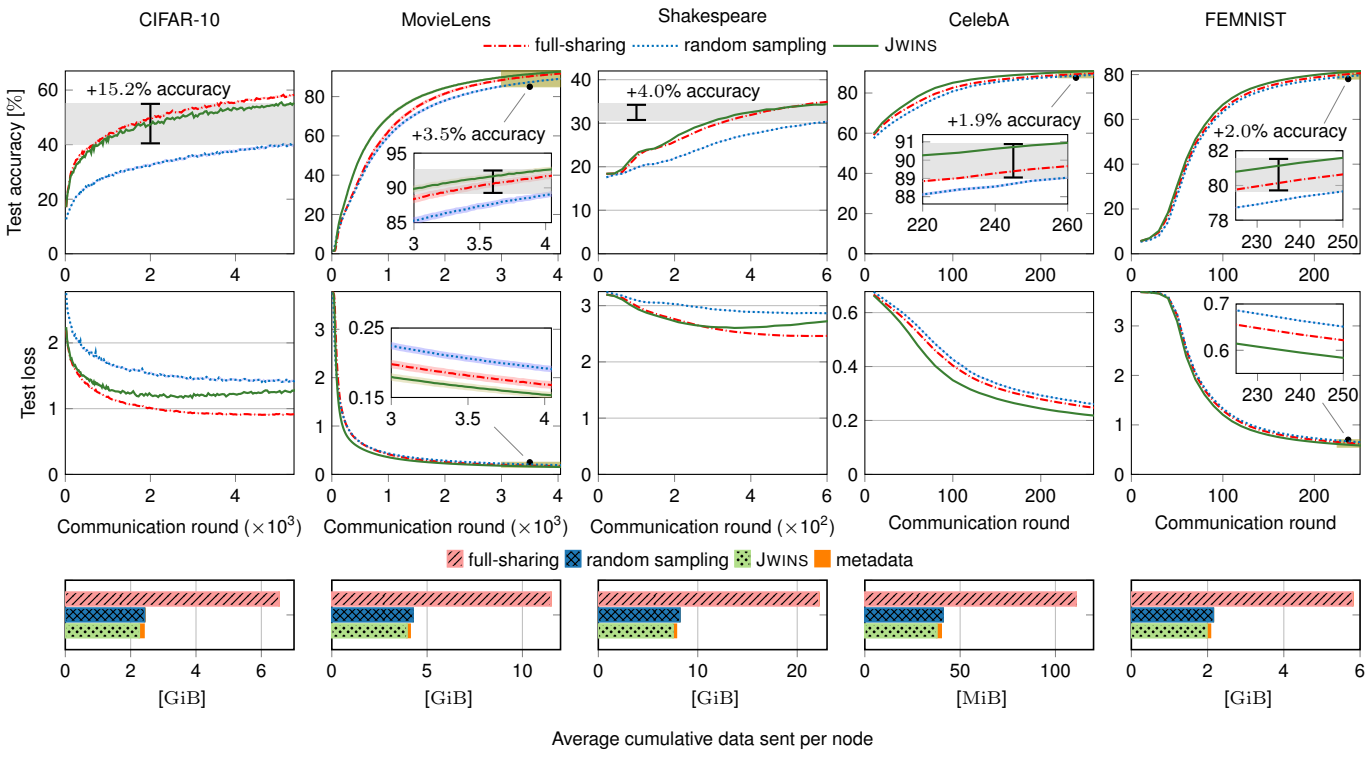

Fig. 4. Learning curves and network usage for JWINS compared to full-sharing and random sampling when run for fixed rounds. JWINS achieves as good test accuracy and test loss as full-sharing across most datasets (row-1 and row-2), while requiring significantly less network transfers per node (row-3). Results are further quantified in Table I.

performance. Note that divergence is not specific to JWINS and full-sharing also begins to diverge if the learning continues.

3) JWINS *vs. random sampling till convergence:* We now run another series of experiments to compare JWINS and random sampling. Effectively, we observe on Figure 4 that random sampling always converges slower than full-sharing and JWINS. However, the longer random sampling is run, the more parameters are synchronized, bringing it close to full-sharing in test accuracy while also significantly increasing the total communication costs. Therefore, when comparing JWINS to random sampling, there are two dimensions to compare against: learning performance for the same number of epochs (Section IV-C), and network costs to reach the same accuracy. We now make a fairer comparison with random sampling in terms of network costs. We run random sampling for even more rounds and measure the best average accuracy reached. We then use this accuracy as the target accuracy for JWINS and full-sharing. Figure 5 shows the number of bytes transferred for full-sharing, random sampling, and JWINS, and their convergence curves for reaching the target accuracy. JWINS always reaches this target accuracy in fewer communication rounds (row-1, Figure 5) than random sampling and therefore pushes between 1.5× to 4× less data on the network depending on the dataset (row-2, Figure 5). Interestingly, the total bytes transferred for CIFAR-10 in random sampling is even larger than full-sharing because of the slow convergence rate of random sampling. Reduced communication rounds also translate to reduced wall-clock time at a similar scale. For instance, in terms of wall-clock time for DL training on CIFAR-10, JWINS took 14 min and random sampling took

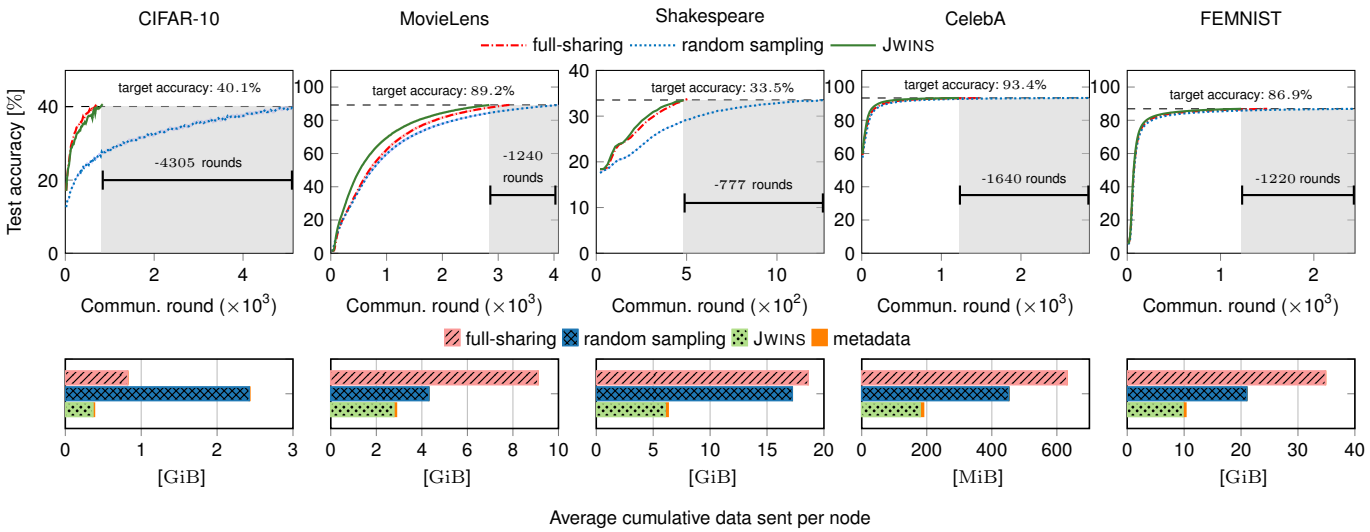

Fig. 5. Learning curves and network usage for JWINS compared to full-sharing and random sampling when run until convergence. In this scenario, the random sampling algorithm is run very long and identified with a target accuracy. Then JWINS and full-sharing are run until this accuracy is reached. JWINS reaches the target accuracy much faster than random sampling (row-1) while requiring 1.5× to 4× less network usage compared to random sampling (row-2).

53 min to reach the same target accuracy (3.7× faster).

D. JWINS *against* CHOCO

CHOCO [6] is a state-of-the-art decentralized learning algorithm based on an error feedback mechanism that can work with multiple communication compression algorithms. We implemented the memory-efficient CHOCO [6] and used TOPK as the compression algorithm since it worked better than random sampling. We challenge both JWINS and CHOCO by limiting the communication budget to 20% and 10% of the communication budget for full-sharing. For the communication budget of 20%, we specify the distribution of α in JWINS as $p_{(\alpha=100\%)} = 0.1$ and $p_{(\alpha=10\%)} = 0.9$, for simplicity. Similarly, for 10% communication budget $p_{(\alpha=100\%)} = 0.05$, $p_{(\alpha=5\%)} = 0.95$. JWINS has the additional advantage of not introducing any new hyperparameters (α distribution concerns the network budget and does not require tuning). CHOCO, instead, introduces an additional hyperparameter: step size (γ), which needs to be tuned separately. In our experiments, we observed that CHOCO is highly sensitive to the choice of γ. We tuned the value of γ to achieve the fastest convergence without crashing the learning ($\gamma_{20\%} = 0.6$ and $\gamma_{10\%} = 0.1$). Given the same communication budget and the exact same setting, we can also fairly compare the runtime performance of JWINS against CHOCO.

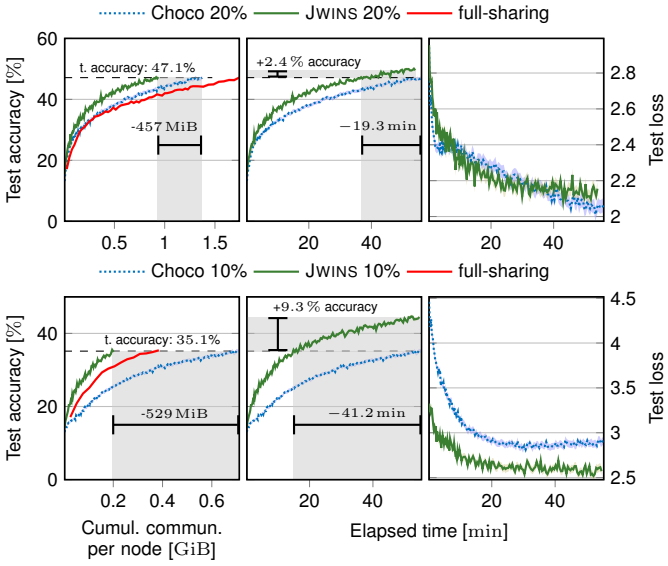

Fig. 6. Performance of JWINS against CHOCO for the communication budget of 20% (row-1) and 10% (row-2). JWINS reaches the target accuracy up to 3.9× faster and achieves up to 9.3% better accuracy for the same communication. The performance gap gets stronger in favor of JWINS as the communication budget gets smaller.

Figure 6 shows the test accuracy against the number of bytes transferred per node to reach the same accuracy, and the convergence plots when run for the same number of rounds for JWINS and CHOCO over the CIFAR-10 dataset. For completeness, we also present the communication volume of full-sharing to reach the target accuracy of CHOCO. The total time taken by JWINS to complete the same number of rounds is slightly less than CHOCO. This means that JWINS is computationally competitive. As full-sharing shares more data in this experiment, we cannot compare the running time of full-sharing against CHOCO and JWINS in the test bed. Even at the optimal values of γ for the learning task, JWINS converges faster than CHOCO, taking up to 3.9× less time and network savings to reach the same accuracy. JWINS outperforms both full-sharing and CHOCO in terms of bytes transferred to reach a target accuracy. As the communication budget decreases, the performance gap increases, with JWINS outperforming CHOCO by 9.3% accuracy. Even with 10% communication

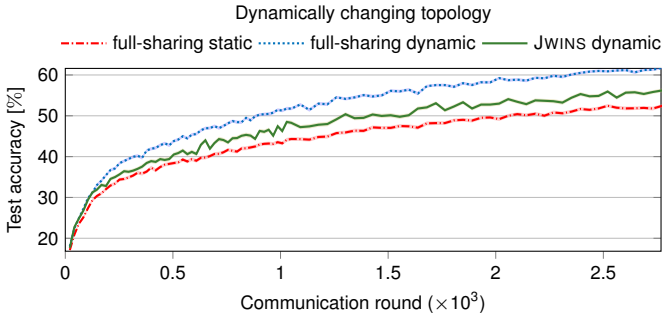

Fig. 7. Randomizing neighbors every round *i.e.*, using a dynamic topology enables faster learning for both full-sharing and JWINS. CHOCO is unsuitable for dynamic topologies, and hence, is not charted here.

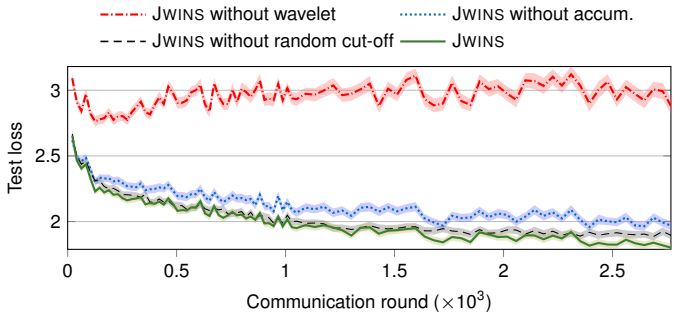

Fig. 8. Ablation study of JWINS on the CIFAR-10 dataset. Removing any of the three components: *(i)* wavelet *(ii)* accumulation or *(ii)* randomized cut-off negatively affects the performance with wavelet being the most significant.

budget, JWINS retains its performance against full-sharing, while CHOCO requires up to 529 MiB less cumulative communication per node to reach the target accuracy.

Another benefit is associated with changing topologies. Through empirical observation, we found that randomizing neighbors at each training round without moving data enables improved performance for full-sharing. This is due to the more effective mixing of models. JWINS demonstrates the same behavior. Since CHOCO works on an error feedback mechanism, it is unsuitable for changing topologies and there is practically no learning in the experimental setup over a wide range of γ. Figure 7 shows the performance of full-sharing in both static and dynamic topologies, as well as JWINS in dynamic topologies with the original α distribution on CIFAR-10. Clearly, dynamic topologies result in better models for full-sharing, and JWINS follows the same trend, performing even better than static-topology full-sharing.

E. *Breaking* JWINS *down*

We now assess how the components of JWINS impact our results in an ablation study. We run a set of experiments by removing only one of the following components of JWINS in each experiment: (1) wavelet transform, (2) accumulation, and (3) randomized sharing strategy. Figure 8 shows the test loss of these experiments on the CIFAR-10 dataset. It can be observed that wavelet transform is one of the most important components of JWINS. Removing wavelet transform significantly degrades the learning performance. On the other hand, removing either accumulation or random communication cut-off causes a similar but less-significant negative impact on the performance of JWINS. When all components are put together, JWINS achieves the minimum test loss.

Needless to say, metadata compression, the final component of JWINS, is important in reducing network utilization. We present the impact of metadata compression in Figure 9 for a small 15-epoch experiment on the CIFAR-10 dataset. The first bar is with no metadata compression, and the second bar is with Elias Gamma compression. Without compression, the metadata is of the same size as the model parameters since both are essentially 32-bit data types. With the Elias Gamma compression, the metadata is compressed by 9.9×, allowing

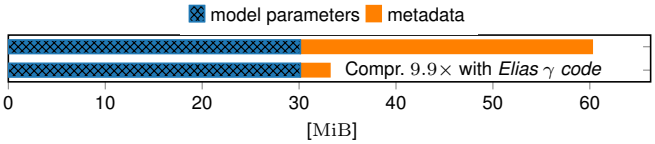

Fig. 9. Size of metadata without and with compression. Without compression, approx. 50% of the communication is wasted.

for more parameter sharing in the same communication budget. This provides clear evidence that each of the components of JWINS is crucial, contributing to the learning performance and network savings.

F. *Scalability of* JWINS

The 96-node decentralized learning is already a moderately large setting. To show that JWINS can be used with a much larger number of nodes, we evaluate the scalability of JWINS on the CIFAR-10 dataset with a less-strict non-IID partitioning (4 shards per node instead of 2). We test JWINS against random sampling in 96-, 192-, 288-, and 384-node settings. The hyperparameters for 192-, 288- and 384-nodes are tuned with the same process described in Section IV-B. We use a 4-regular graph for 96 nodes, 5-regular graph for 192 and 288 nodes, and 6-regular graph for 384 nodes so that the number of edges grows in proportion to the new nodes, preventing very sparsely connected graphs. The number of training samples per node also declines linearly with the addition of nodes since the same dataset is now partitioned across more nodes.

Figure 10, row-1 depicts the evolution of test accuracy in these settings with different nodes. We observe that JWINS consistently achieves better accuracies compared to random sampling, also at a faster convergence rate in all scenarios. We can also observe that JWINS does not compromise on the final accuracy, even though each node has a smaller dataset as the number of nodes increases. In fact, the gross network savings by JWINS to reach a target accuracy also grow with the addition of nodes since each node now shares more data, observed in row-2 of Figure 10 from left to right.

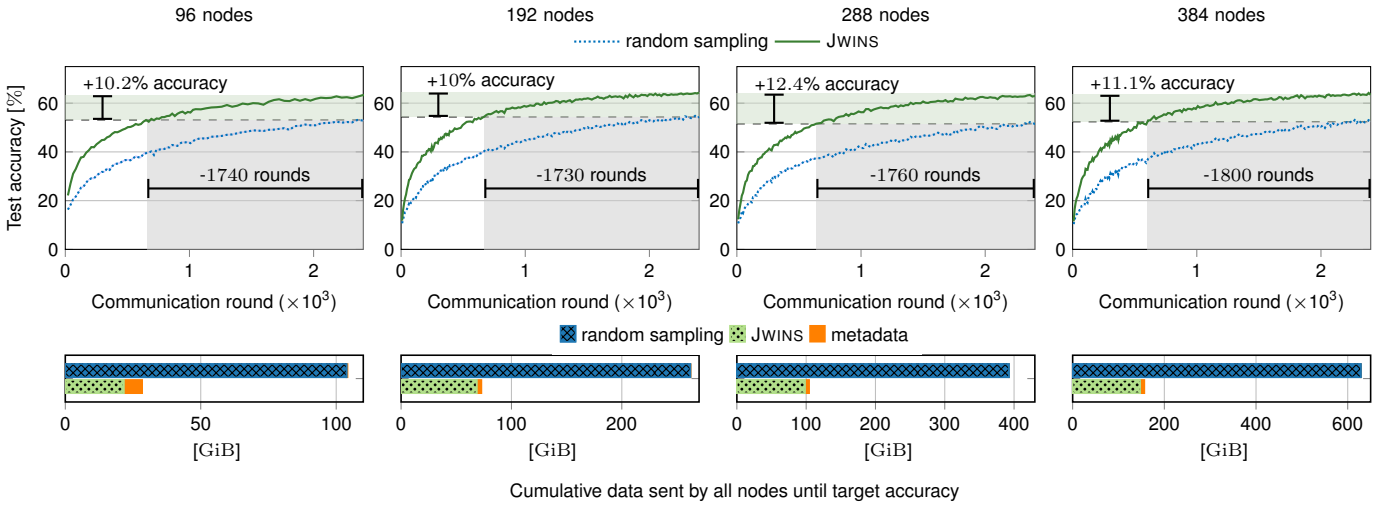

Fig. 10. Scalability study for JWINS on the CIFAR-10 dataset. The number of nodes is increased from 96 by $2\times, 3\times$, and $4\times$. JWINS continues to achieve higher test accuracy than random sampling (row-1). In fact, with the addition of new nodes, the gross network savings of JWINS amplify compared to random sampling as more data is now exchanged (row-2, left to right). Note that the data partitioning here is less-strict non-IID, hence making the values different from the ones presented in the main text.

G. Discussion

a) Learning performance and network savings: In the presence of large deep learning models and also a large number of communication rounds, full-sharing results in each node sharing up to 22 GiB during the whole training process, as shown in Figure 4. This makes full-sharing a highly communication-inefficient solution, despite working very well for decentralized learning tasks.

Tackling the disadvantage of full-sharing in terms of network load, JWINS shares a subset of model parameters with the neighbors. JWINS saves up to 60% in network usage, even though it adds some metadata to the messages for positional correspondence when transferring sparse vectors.

If we would allow the training to continue for an unbounded number of rounds, full-sharing would eventually achieve an accuracy slightly better than JWINS. In real-life decentralized learning tasks, this is however impractical.

b) Flexibility of JWINS*:* In this paper, we present JWINS as a system composed of multiple modules. However, it is worth highlighting that the wavelet-based parameter ranking component of JWINS holds potential for standalone use and could be integrated into other sparsification algorithms. Furthermore, the parameter compression (Fpzip) across all experiments showed that JWINS is compatible with general-purpose compression algorithms. Finally, JWINS works across a wide variety of models (CNN, LSTM, Embeddings, FC) without requiring any changes, since JWINS considers models as flat vectors of parameters.

V. Related work

Communication compression is a hot topic of research in distributed learning because of the large size of deep neural networks currently used. To the best of our knowledge, JWINS is the first work to use sparsification over wavelet parameter ranking and accumulation in combination with a randomized sharing strategy.

a) Communication compression in fully decentralized learning: In DL, quantization along with compression over the change of the model in a round has been used [8] with convergence guarantees for unbiased compression algorithms. On similar lines, CHOCO-SGD [6], [7] proposes theoretical results and convergence guarantees for a wider class of compression algorithms. In both of these works, each node stores a replica of models of all other nodes to allow sharing of the compressed model-change. Note that a memory-efficient version of CHOCO-SGD has also been proposed [6]. POWERGOSSIP uses low-rank linear compressors applied to model difference for communication efficiency and it performs as good as CHOCO-SGD without introducing any hyperparameter. Our work differs from these in four ways: (1) nodes in JWINS do not maintain replicas of models from others, making JWINS more memory-efficient, and flexible to nodes leaving and joining, (2) we propose a combination of the parameter ranking algorithm using wavelets and a random cut-off based parameter strategy (here, communication compression), (3) JWINS is shown to work with a much larger number of nodes (Section IV-F), and (4) we evaluate JWINS from a systems perspective in the presence of difficult non-IID data distributions.

SAPS-PSGD [27] combines the sparsification of random sampling with an adaptive peer-selection scheme for reducing the communication in decentralized learning with independent and identically distributed (IID) data. Partitioning schemes for logistic regression models into consecutive blocks of a given size have been proposed over random sampling. In contrast, our focus lies on non-IID data partitioning and deep neural networks, an approach that mirrors real-world scenarios more closely and also presents a greater degree of challenge.

Moreover, JWINS does not assume anything about the topology of the nodes, therefore can be combined with peer-sampling and selection services (Section IV-D). This is an interesting avenue for future research.

b) Compression in distributed learning: Distributed learning with all-reduce synchronizations every round, such as the parameter server architecture, provides a much simpler platform for communicating less because every node has the same model at the start of each training round. This allows nodes to share only gradients instead of the actual parameters. GAIA [15] accumulates the local gradients in an accumulation vector and only shares them once they are higher than a certain threshold. DSSGD [10] empirically shows that the gradients can be sparsified by up to 2 orders of magnitude in parameter-server settings using TOPK. Alistarh et al. [12] proved the convergence of TOPK gradient sharing if certain assumptions are satisfied. TOPK gradient sharing is also used in DEEP GRADIENT COMPRESSION [13] along with accumulation to account for parameters that change slowly, while also using momentum correction to handle stale accumulated updates. SKEWSCOUT [14] builds on top of these communication compression techniques, like GAIA [15] and DEEP GRADIENT COMPRESSION [13], by adaptively setting their hyperparameters.

All these works either rely on a central coordinator or on a global model synchronization. In contrast, JWINS is designed to work in a fully-decentralized manner, where local models of each node are not globally synchronized.

c) Wavelet transform in model compression: Wavelet transform has been mentioned as means of data compression for ML purposes [42], [43]. However, to the best of our knowledge, it was never used. JWINS actually implements DWT for decentralized learning to represent the importance of parameters and wavelet proves to be quite effective.

VI. CONCLUSION

Overparameterized machine learning models and a large number of communication rounds in decentralized learning cause the network to be the bottleneck. In this paper, we present JWINS, a system for communication-efficient decentralized learning. JWINS maintains learning performance while reducing communication by using a smart sparsification technique, dictated by the parameter ranking and selection schemes of JWINS in the wavelet-frequency domain.

Our evaluations across 5 datasets and 3 different learning tasks show that JWINS can reduce the data transferred in DL by up to 64% compared to full-sharing without deteriorating the performance. Further, JWINS achieves the convergence accuracy of its random sampling competitor significantly faster while transferring up to $4\times$ fewer bytes. We also show that the parameter ranking in JWINS is highly effective through an ablation study.

Interesting avenues of future research can be along both dimensions of parameter ranking and parameter selection. An adaptive version of the importance score based on the parameter type (CNN, RNN, FC) may be explored in depth. Theoretical convergence guarantees of JWINS would complement our empirical evaluations. Parameter selection based on inferred knowledge about the models of other participating nodes is another future research direction.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*, 2015. arXiv: 1502.01852.

[2] A. Hannun, C. Case, J. Casper, *et al.*, *Deep speech: Scaling up end-to-end speech recognition*, 2014. arXiv: 1412.5567.

[3] W. He, X.-Y. Zhang, F. Yin, and C.-L. Liu, "Deep direct regression for multi-oriented scene text detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 745–753. DOI: 10.1109/ICCV.2017.87.

[4] X. Lian, C. Zhang, H. Zhang, C.-J. Hsieh, W. Zhang, and J. Liu, "Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent," in *NIPS*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf.

[5] A. Koloskova, N. Loizou, S. Boreiri, M. Jaggi, and S. Stich, "A unified theory of decentralized SGD with changing topology and local updates," in *ICML*, 2020. [Online]. Available: https://proceedings.mlr.press/v119/koloskova20a.html.

[6] A. Koloskova, S. Stich, and M. Jaggi, "Decentralized stochastic optimization and gossip algorithms with compressed communication," in *ICML*, 2019. [Online]. Available: https://proceedings.mlr.press/v97/koloskova19a.html.

[7] A. Koloskova, T. Lin, S. U. Stich, and M. Jaggi, "Decentralized deep learning with arbitrary communication compression," in *ICLR*, 2020. [Online]. Available: https://openreview.net/pdf?id=SkgGCkrKvH.

[8] H. Tang, S. Gan, C. Zhang, T. Zhang, and J. Liu, "Communication compression for decentralized training," in *NeurIPS*, 2018. [Online]. Available: https://papers.nips.cc/paper_files/paper/2018/file/44feb0096faa8326192570788b38c1d1-Paper.pdf.

[9] T. Vogels, S. P. Karimireddy, and M. Jaggi, "Practical low-rank communication compression in decentralized deep learning," in *NeurIPS*, 2020. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/a376802c0811f1b9088828288eb0d3f0-Paper.pdf.

[10] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2015, pp. 909–910. DOI: 10.1109/ALLERTON.2015.7447103.

[11] N. Strom, "Scalable distributed DNN training using commodity GPU cloud computing," in *16th Annual Conference of the International Speech Communication Association*, 2015. [Online]. Available: https://www.isca-speech.org/archive_v0/interspeech_2015/papers/i15_1488.pdf.

[12] D. Alistarh, T. Hoefler, M. Johansson, S. Khirirat, N. Konstantinov, and C. Renggli, "The convergence of sparsified gradient methods," in *NeurIPS*, Montréal, Canada, 2018. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2018/file/314450613369e0ee72d0da7f6fee773c-Paper.pdf.

[13] Y. Lin, S. Han, H. Mao, Y. Wang, and B. Dally, "Deep gradient compression: Reducing the communication bandwidth for distributed training," in *ICLR*, 2018. [Online]. Available: https://openreview.net/forum?id=SkhQHMW0W.

[14] K. Hsieh, A. Phanishayee, O. Mutlu, and P. B. Gibbons, "The non-IID data quagmire of decentralized machine learning," in *ICML*, 2020. [Online]. Available: http://proceedings.mlr.press/v119/hsieh20a/hsieh20a.pdf.

[15] K. Hsieh, A. Harlap, N. Vijaykumar, *et al.*, "Gaia: Geo-Distributed machine learning approaching LAN speeds," in *NSDI*, 2017. [Online]. Available: https://www.usenix.org/conference/nsdi17/technical-sessions/presentation/hsieh.

[16] A. Krizhevsky, V. Nair, and G. Hinton, "The cifar-10 dataset," vol. 55, no. 5, 2014. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html.

[17] F. M. Harper and J. A. Konstan, "The movielens datasets: History and context," *ACM Trans. Interact. Intell. Syst.*, vol. 5, no. 4, 2015. DOI: 10.1145/2827872.

[18] S. Caldas, S. M. K. Duddu, P. Wu, *et al.*, *Leaf: A benchmark for federated settings*, 2019. arXiv: 1812.01097.

[19] R. Ormándi, I. Hegedűs, and M. Jelasity, "Gossip learning with linear models on fully distributed data," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 4, 2013. DOI: 10.1002/cpe.2858.

[20] Y. Collet, *LZ4*, 2022. [Online]. Available: http://www.lz4.org/.

[21] I. Pavlov, *LZMA SDK*, 2022. [Online]. Available: https://www.7-zip.org/.

[22] F. Seide, H. Fu, J. Droppo, G. Li, and D. Yu, "1-bit stochastic gradient descent and application to data-parallel distributed training of speech DNNs," in *Interspeech 2014*, 2014. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/IS140694.pdf.

[23] D. Alistarh, D. Grubic, J. Z. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *NIPS*, 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/6c340f25839e6acdc73414517203f5f0-Paper.pdf.

[24] M. Li, D. G. Anderson, J. W. Park, *et al.*, "Scaling distributed machine learning with the parameter server," in *OSDI*, 2014. [Online]. Available: https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-li_mu.pdf.

[25] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *AISTATS*, 2017. [Online]. Available: https://proceedings.mlr.press/v54/mcmahan17a.html.

[26] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *2017 Conference on Empirical Methods in Natural Language Processing*, 2017. DOI: 10.18653/v1/D17-1045.

[27] Z. Tang, S. Shi, and X. Chu, "Communication-efficient decentralized learning with sparsification and adaptive peer selection," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020. DOI: 10.1109/ICDCS47774.2020.00153.

[28] I. Hegedűs, G. Danner, and M. Jelasity, "Decentralized learning works: An empirical comparison of gossip learning and federated learning," *Journal of Parallel and Distributed Computing*, vol. 148, pp. 109–124, 2021. DOI: 10.1016/j.jpdc.2020.10.006.

[29] T. Mantoro and F. Alfiah, "Comparison methods of DCT, DWT and FFT techniques approach on lossy image compression," in *2017 International Conference on Computing, Engineering, and Design (ICCED)*, 2017, pp. 1–4. DOI: 10.1109/CED.2017.8308126.

[30] M. Ramkumar and G. Anand, "An FFT-based technique for fast fractal image compression," *Signal processing*, vol. 63, no. 3, 1997. DOI: 10.1016/S0165-1684(97)00162-X.

[31] R. Patel, S. Katiyar, and K. Arora, "An improved image compression technique using huffman coding and FFT," in *International conference on smart trends for information technology and computer communications*, Springer, 2016. DOI: 10.1007/978-981-10-3433-6_7.

[32] D. S. Taubman and M. W. Marcellin, "JPEG2000: Standard for interactive imaging," *Proceedings of the IEEE*, vol. 90, no. 8, pp. 1336–1357, 2002. DOI: 10.1109/JPROC.2002.800725.

[33] P. Elias, "Universal codeword sets and representations of the integers," *IEEE transactions on information theory*, vol. 21, no. 2, pp. 194–203, 1975. DOI: 10.1109/TIT.1975.1055349.

[34] A. Dhasade, A.-M. Kermarrec, R. Pires, R. Sharma, and M. Vujasinovic, "Decentralized learning made easy with DecentralizePy," in *3rd Workshop on Machine Learning and Systems*, 2023. DOI: 10.1145/3578356.3592587.

[35] A. Paszke, S. Gross, F. Massa, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *NeurIPS*, 2019. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[36] L. Boccassi *et al.*, *Zeromq: An open-source universal messaging library*, 2022. [Online]. Available: https://zeromq.org.

[37] G. Lee, R. Gommers, K. Wohlfahrt, *et al.*, *Pywavelets/pywt: Pywavelets 1.3.0*, version v1.3.0, Mar. 2022. DOI: 10.5281/zenodo.6347505.

[38] A. Dhasade, N. Dresevic, A.-M. Kermarrec, and R. Pires, "TEE-based decentralized recommender systems: The raw data sharing redemption," in *36th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022. DOI: 10.1109/IPDPS53621.2022.00050.

[39] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, pp. 30–37, 2009. DOI: 10.1109/MC.2009.263.

[40] L. Xiao and S. Boyd, "Fast linear iterations for distributed averaging," *Systems & Control Letters*, vol. 53, no. 1, pp. 65–78, 2004. DOI: 10.1016/j.sysconle.2004.02.022.

[41] P. Lindstrom and M. Isenburg, "Fast and efficient compression of floating-point data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 1245–1250, 2006. DOI: 10.1109/TVCG.2006.143.

[42] S. Jin, S. Di, X. Liang, J. Tian, D. Tao, and F. Cappello, "DeepSZ: A novel framework to compress deep neural networks by using error-bounded lossy compression," in *28th International Symposium on High-Performance Parallel and Distributed Computing*, 2019. DOI: 10.1145/3307681.3326608.

[43] T. Cassimon, S. Vanneste, S. Bosmans, S. Mercelis, and P. Hellinckx, "Designing resource-constrained neural networks using neural architecture search targeting embedded devices," *Internet of Things*, vol. 12, p. 100 234, 2020. DOI: 10.1016/j.iot.2020.100234.