EVALUATION OF MODEL COMPRESSION TECHNIQUES IN FEDERATED LEARNING UNDER BANDWIDTH CONSIDERATIONS

TECHNICAL REPORT

Vaishnavi Jeurkar
School of Computing
Newcastle University
Newcastle Upon Tyne, NE4 5TG
v.s.jeurkar2@ncl.ac.uk

Dr. Peter Michalak
Senior Data Scientist
National Innovation Center for Data
Newcastle Upon Tyne, NE4 5TG
Peter.Michalak@ncl.ac.uk

Dr. Jacek Cala
Senior Data Scientist
National Innovation Center for Data
Newcastle Upon Tyne, NE4 5TG
Jacek.Cala@ncl.ac.uk

August 12, 2024

ABSTRACT

Federated learning offers a promising approach for edge training, but its deployment is hindered by the limited bandwidth and computing resources of edge devices. Existing research presents various model compression techniques, including quantization, pruning, and knowledge distillation, to address these challenges. This paper investigates the application of structured and unstructured pruning as model compression algorithms within the context of federated learning, using the Flower framework for simulation. Our approach involves dynamically selecting and pruning weights or neurons at each training round on the client side, followed by sparse matrix communication, thereby reducing the data transferred to the server. We specifically target scenarios with bandwidth-constrained devices that can train models over multiple rounds but where the amount of data transferred, and consequently the duration of wireless transmission, is limited. Extensive experiments demonstrate that our proposed pruning and sparsification approach significantly reduces upstream communication bytes while maintaining accuracy, albeit with an increased number of rounds. For instance, training a ResNet-12 model on the MNIST dataset achieves 97% accuracy in 15 rounds, requiring 155.74 megabytes of data transfer per round whereas, an 80% pruned model reaches the same accuracy in the same number of rounds, with only 49.35 megabytes transferred per round. This research addresses deployment scenarios with limited bandwidth by comparing two model compression techniques and examining their impact on accuracy and the number of training rounds. It also raises important questions regarding the optimal model size for specific deep learning tasks and the necessity of large complex models for achieving peak accuracy.

Keywords Federated Learning · Pruning · Model Compression · Image Classification · Flower

1 Introduction

Federated Learning (FL) has emerged as a powerful paradigm for training machine learning models while preserving data privacy. It allows distributed training on data residing on user devices, eliminating the need for direct data sharing with a central server. [1] This is particularly attractive for applications processing sensitive data, such as healthcare or finance. However, the communication overhead associated with FL can be significant, especially for bandwidth-constrained devices. Client devices in FL are usually much more resource-constrained than server machines in terms of computation power, communication bandwidth, memory, and storage size. For example, in a smart healthcare scenario, where wearable devices collect real-time health metrics such as heart rate, blood pressure, or glucose levels, the need to frequently communicate large model updates can quickly overwhelm the limited bandwidth available, potentially leading to delays that could compromise timely interventions. Another scenario involves drones deployed in disaster response operations; these drones need to collaboratively learn and share data about environmental conditions and obstacles. Given their reliance on limited battery life and wireless communication in often harsh environments, minimizing the communication overhead is critical to ensure they can function effectively without exhausting their

power reserves or losing connection. To address this challenge, various model compression techniques have been developed to reduce the size of the transmitted updates, thereby enhancing communication efficiency.

In standard FL, the server sends a model to a group of clients, who train it on their local data and then send their updated parameters back to the server for aggregation. The aggregated model parameters are then used for client and server evaluation, this is one round of FL. A comprehensive survey of several model aggregation techniques like FedProx [2], FedNova [3], FedAvg [1], etc., for optimal parameter aggregation is provided in [4]. FedAvg is widely used due to its simplicity and effectiveness, it performs well across diverse datasets and network conditions, making it a versatile choice for various federated learning scenarios. There are existing open-source frameworks for federated learning, like TensorFlow Federated [37], PySyft, OpenFL, etc. Flower [8] was chosen for experimentation due to its ease of use and widespread adoption in the edge AI domain. Flower's simulation feature, VirtualClientEngine (VCE), enables the simulation of federated learning workloads across various scenarios without the need for extensive physical device management. This feature is particularly useful for running workloads quickly on available compute systems, validating algorithms under different conditions, and accommodating varying levels of data and system heterogeneity, client availability, and privacy budgets. Two image classification datasets, MNIST and CIFAR-10, were selected for this project [9]. These datasets are frequently used in literature, facilitating easy comparison of model compression results with other research. PyTorch [36] was chosen for model development due to its flexibility and robust support for various deep learning tasks.

Model pruning is an extensively studied model compression technique that removes unimportant parameters from a deep learning model, this helps in reducing the model size and enables efficient inference [5]. Post-training pruning involves pruning the model after it has been fully trained, identifying and removing less important connections, neurons, or entire structures. There are two main types of pruning: structured pruning, which removes entire structures like neurons or filters, and unstructured pruning, which removes individual weight parameters [see figure 2]. Additionally, pruning can be local, affecting individual layers, or global, considering the overall importance across the entire network. Because models can be over-parameterized, there are many redundancies that can be exploited to improve efficiency, making pruning a highly successful method in centralized training settings. [6]

This paper proposes an efficient federated algorithm that implements model pruning using the PyTorch pruning library [7] within the Flower framework. It compares the performance of local structured pruning and global unstructured pruning to reduce the number of bytes transferred during upstream communication. The main contributions of this paper are:

1. **Performance Comparison**: We compare the performance of two types of pruning: local structured pruning and global unstructured pruning by pruning the parameters on the client side after training during each round. We conducted extensive experiments to identify patterns in model compression performance using the CIFAR-10 and MNIST datasets. To validate the effectiveness of our proposed framework, we employed two model architectures across the datasets: SimpleCNN and ResNet12 for the MNIST dataset, and ResNet12 for the CIFAR-10 dataset.
2. **Pruning and Sparse Matrix Communication**: The paper combines pruning with sparse matrix communication, reducing the parameter size during communication while maintaining the model structure by reconstructing the model parameters at the server. This approach minimizes errors caused by pruning and structural changes to the model.
3. **Model size optimization for Bandwidth-Constrained Devices**: These experiments determine the optimal number of rounds needed to achieve a target accuracy while minimizing the size of model parameters transferred, which is crucial for bandwidth-constrained devices. They also reveal instances where excessive model pruning results in an irrecoverable loss of accuracy because the model becomes too small.

2 Literature Review

Federated Learning often faces challenges related to communication overhead, especially when dealing with bandwidth-constrained devices. To address this, various model compression techniques have been developed to enhance communication efficiency without significantly compromising model performance. Pruning and sparsification techniques aim to reduce the size and complexity of neural networks to enhance communication efficiency. Pruning removes specific parameters or structures deemed less important, while sparsification focuses on reducing the number of non-zero elements in tensors. Despite their different approaches, both techniques seek to minimize the amount of data transmitted during FL, thereby addressing the bandwidth constraints of edge devices.

A convex optimization framework [14] aims to minimize gradient coding length and achieve communication efficiency, investigating a range of techniques to optimize the trade-off between sparsity (reducing transmitted elements) and model performance. A sparse ternary compression technique [15] extends the existing top-k gradient sparsification

technique with a mechanism that enables both downstream compression and ternarization, along with optimal Golomb encoding of the weight updates. Another compression algorithm [16] employs three procedures: model sparsification, parameter clustering, and perturbed compression. Sparsification filters out redundant parameters, clustering enhances parameter correlation, and perturbed compression reshapes and compresses parameters using a compression matrix. DeepReduce sparsification [17] efficiently compresses sparse tensors into values and indices with techniques like curve fitting and bloom filters, reducing communication overhead without sacrificing training accuracy.

PruneFL [18] proposes using unstructured pruning during training and supports corresponding sparse matrix computations by extending the deep learning framework. This approach, which utilizes dense matrices for full-sized models and sparse matrices for weights in both convolutional and fully connected layers of pruned models, faces challenges in generalization. In contrast, our method does not involve sparse matrix computations. Different pruning techniques, such as iterative pruning [28], soft pruning [29], and dynamic pruning [30], have been developed to identify redundant parameters during training. Federated Pruning [6], on the other hand, performs structural pruning on the server and sends the reduced model to the clients for training. Several studies [19], [20], [21], [22], [23], [24], [25], [26], [27] have introduced various criteria for assessing the importance of convolution kernels in structured pruning, leading to the elimination or nullification of redundant kernels. Similar work to ours has been done in [31], where unstructured pruning and quantization were implemented separately within the Flower framework. However, our approach advances this by incorporating the PyTorch pruning library into Flower, enhancing the code's adaptability for a broader range of applications.

Quantization, a widely explored technique, reduces the number of bits required to represent model parameters, significantly shrinking communication costs. Techniques like Quantized Stochastic Gradient Descent (QSGD) [10] compress gradients during communication in distributed training, reducing bandwidth usage at the expense of potentially increased training time. QSGD outperforms various quantization schemes, such as SignSGD [11] and TernGrad [12], and its Bayesian variant, QLSD-Bayes [13], demonstrates superior performance within the Bayesian Federated Learning setting. Hybrid methods combine multiple compression techniques to achieve better overall performance. For instance, a three-step compression algorithm [32] integrates pruning, quantization, and Huffman coding to optimize communication efficiency. Another algorithm [33] introduces two forms of compression: downstream compression on the server side after federated averaging, and upstream compression on the client side, which includes training a mask or initializing a smaller model using a saliency criterion. Additionally, these methods quantize the non-zero elements of the model to 8-bit width using a low-precision quantizer. Another hybrid approach [34] combines sparsification, quantization, and encoding to efficiently represent the sparsified and quantized weights for transmission. Among the Flower framework's baselines, DepthFL [35] is a knowledge distillation approach that uses federated learning with depth scaling. It creates local models of different depths based on clients' resources. DepthFL improves global model performance better than width scaling by enabling deep layers to be trained through mutual self-distillation among classifiers of varying depths within a local model.

Various model compression techniques have been explored to address the communication challenges in FL. Pruning, sparsification, quantization, knowledge distillation, and hybrid methods each offer unique advantages and can be tailored to specific FL scenarios to enhance efficiency while maintaining model performance. This paper focuses on exploring pruning along with sparse matrix communication as a model compression technique within the Flower framework, comparing local structured pruning and global unstructured pruning to identify the most effective approach for bandwidth-constrained devices.

3 Methodology

Initial efforts were focused on implementing model compression for object detection using the VisDrone dataset [38], as minimizing communication overhead is crucial for drones to operate efficiently without draining their power reserves or losing connection. However, due to the complexity of object detection tasks and the need for a thorough examination of pruning effects on models, we opted to use simpler and more commonly used datasets for our model compression experiments. This approach enabled us to effectively compare our results with existing methods.

Federated learning, evaluation, and analytics require infrastructure to handle model transfer, local training, and aggregation of updated models. Flower provides a scalable and secure infrastructure to facilitate these tasks, offering a unified approach to federated learning, analytics, and evaluation. In this project, Flower simulation was employed to create a federated environment. To simulate a typical federated learning scenario, 100 clients were initialized, each with distinct training and test sets. For each training and evaluation round, two clients were randomly selected to participate. In each round, model parameters received from server are first set to the client's model, followed by training on the client's training set. Post-training, the model is pruned using either unstructured or structured pruning methods. These pruned model parameters are converted to sparse matrices before sending them back to the server for

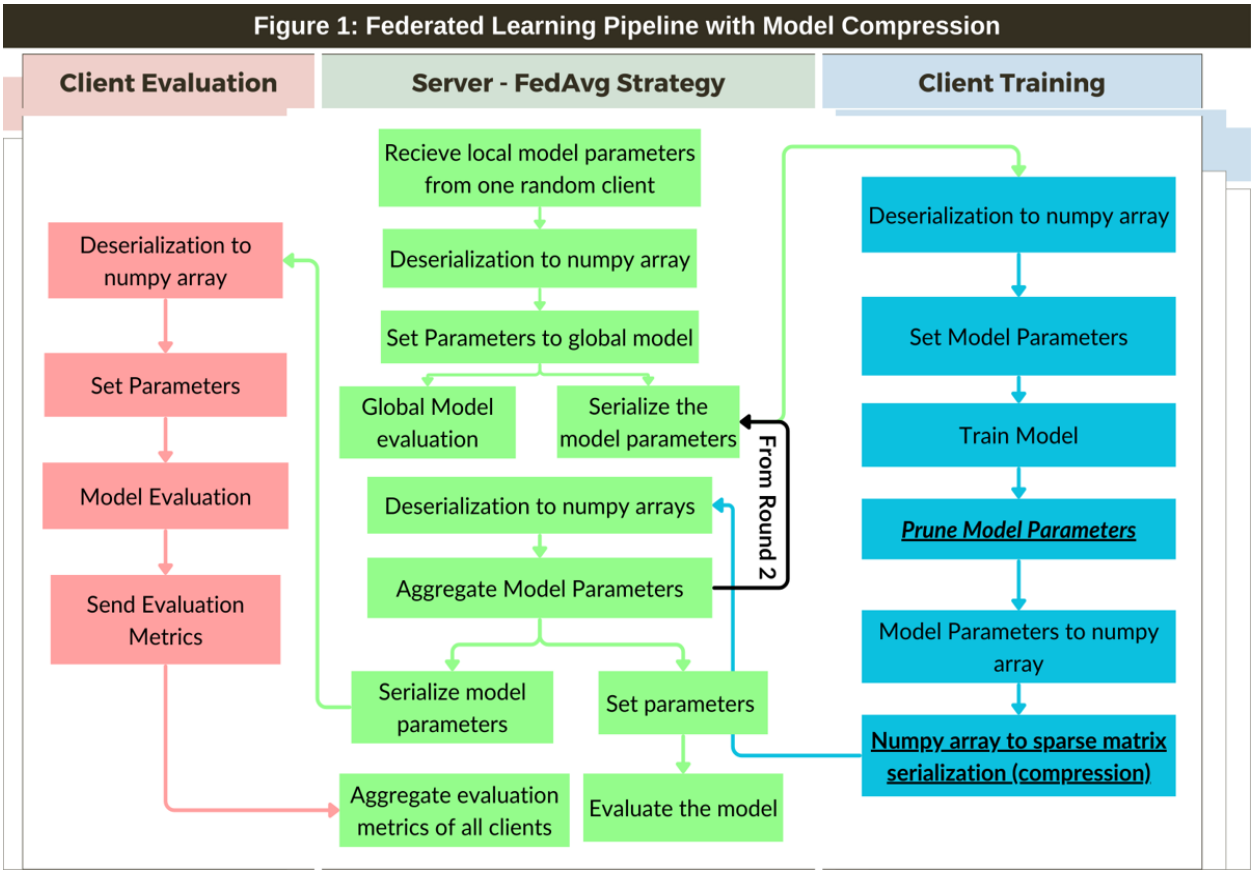

Figure 1: Federated Learning Pipeline with Model Compression

aggregation. Converting the parameters to sparse matrices reduces the number of bytes required for communication. FedAvg algorithm is used for model parameter aggregation, it operates by averaging the model parameters across multiple clients to produce a global model. These aggregated model parameters are then used for client and server side evaluations. Figure 1, displays the flowchart for federated learning pipeline with model compression.

To optimize the federated learning process, the Flower Client class was utilised to build clients. The Client class required handling parameter serialization and deserialization, which was managed by converting model parameters into sparse matrices. A custom strategy was developed to handle the serialization and deserialization process, while other functionalities were inherited from FedAvg. [1]

3.1 Dataset Distribution

In this project, two datasets were utilized: CIFAR-10 and MNIST [9]. The CIFAR-10 dataset comprises 60,000 32x32 color images categorized into 10 classes, with each class containing 6,000 images. The dataset is divided into 50,000 training images and 10,000 test images. The MNIST dataset, a staple for handwritten digit classification tasks, consists of 70,000 labeled 28x28 pixel grayscale images of handwritten digits, split into 60,000 training images and 10,000 test images, across 10 classes representing digits 0 through 9. As shown in Figure 3, for both datasets, the training and test sets were loaded separately. The training set was further partitioned into 100 clients, with each client's data split into 90% for training and 10% for validation, used for client-side evaluation. The centralized test set was reserved for evaluation at the server.

3.2 Model Architecture

The ResNet12 model is a complex deep neural network leveraging residual blocks to enable efficient training of deep architectures by addressing vanishing gradient issues, featuring multiple layers with extensive use of batch normalization

[1] https://github.com/vaishnavijeurkar/Evaluation-of-Model-Compression-Techniques-in-Federated-Learning-

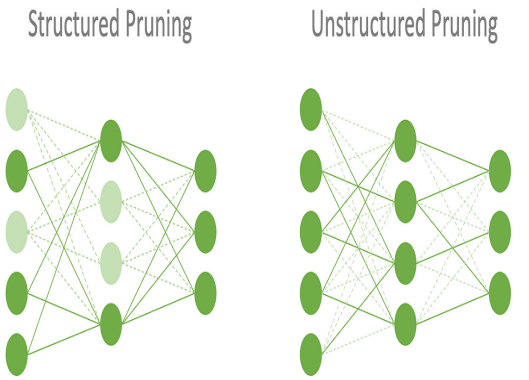

Figure 2: Comparison of Structured and Unstructured Pruning.

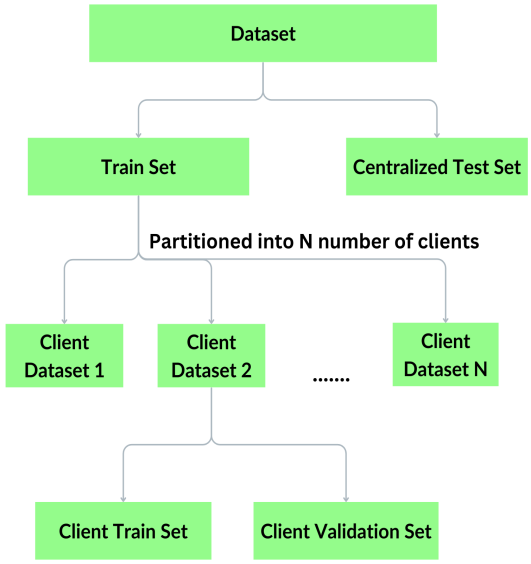

Figure 3: Data Partitioning in Flower

and convolutional operations. In contrast, the Simple CNN model exemplifies a more straightforward architecture, ideal for less complex tasks, with only two convolutional layers followed by pooling and fully connected layers, making it significantly simpler and less computationally intensive than ResNet12.

The ResNet12 model was implemented on both the MNIST and CIFAR-10 datasets, while the Simple CNN model was implemented only on the MNIST dataset. This approach was taken to evaluate whether the results of our pruning strategy are consistent and comparable across different model complexities and datasets.

3.3 Pytorch Pruning

In this project, pruning was implemented using the PyTorch pruning library, `torch.nn.utils.prune`, which effectively nullifies the pruned parameters of neural networks and allows implementation of custom pruning techniques. The pruning process involves several steps, beginning with the selection of a pruning technique followed by specifying the module and the parameter within that module to prune. When pruning a model using this library, additional parameters are created: `weight_orig` representing the unpruned version of the weights, and `weight_mask` representing the mask applied to these weights to obtain the pruned weights. To finalize pruning and make it permanent, the reparameterization is removed by multiplying the weights with their corresponding mask.

For this study, global unstructured pruning was implemented using the `L1Unstructured` method. This method prunes individual connections based on the $L1$-norm of the weights, effectively removing the weights with the smallest absolute values across the entire network. This technique allows for a fine-grained pruning approach, creating a highly sparse network by eliminating the least significant weights. The use of global unstructured pruning ensures that the sparsity is distributed throughout the model, potentially improving the overall efficiency without severely impacting the model's performance.

In contrast to unstructured pruning, structured pruning was applied to each layer of the neural networks. Structured pruning removes entire channels or filters based on the Ln-norm along a specified dimension, thus simplifying the network architecture. However, it is important to note that PyTorch currently does not support global structured pruning, global pruning is removing weights across the entire network instead of individual layers. Local pruning is often applied to individual layers, meaning each layer has its own criterion for pruning, rather than applying a global criterion across the entire network. Despite this limitation, structured pruning within layers effectively reduces the model complexity while retaining its representational capacity.

3.4 Experimental Setup

The MNIST and CIFAR-10 datasets were chosen for their prevalence in the literature, which facilitates straightforward comparison of our model compression results with previous research. A total of six experiments were conducted, exploring various combinations of model architectures and datasets. Specifically, the experiments involved the MNIST dataset with the ResNet12 and SimpleCNN models, and the CIFAR-10 dataset with the ResNet12 model. Each model was subjected to both local structured and global unstructured pruning strategies. The distribution of datasets followed an Independent and Identically Distributed (IID) nature, ensuring uniformity in data distribution across clients. The experiments varied in the number of training rounds required for convergence, reflecting the different convergence characteristics of each model and pruning strategy. Each model was trained for one epoch per round on individual clients, with 2% of clients selected for training in each round.

In all experiments, clients utilized the Stochastic Gradient Descent (SGD) optimizer with a learning rate of 0.01 and a momentum of 0.9. The ResNet12 experiments, specifically, were executed on a GPU to leverage its computational capabilities for faster training and evaluation. The number of bytes for each round is calculated before the sparse matrix is communicated to the server. This setup ensured that the models were efficiently trained and evaluated, allowing for a thorough investigation into the effectiveness of different pruning strategies across various architectures and datasets.

3.5 Implementation Challenges

An implementation challenge arose due to the structural changes in the model's state dictionary following the application of pruning. After removing reparameterization during structured pruning, the order of parameters in the state dictionary changes to bias-weight-bias instead of the original weight-bias-weight order. This reordering caused errors when loading the state dictionary at the server for aggregation. In the Flower framework, the model's state dictionary is converted to a `NumPy` array and then serialized for communication. To address this issue, a new model is initialized, and the pruned model's state dictionary is first loaded into the new model to correct the order. Subsequently, the new model's state dictionary is communicated to the server, ensuring consistency and correctness in parameter transmission.

Pruning the model using the PyTorch library does not reduce the model's size, as it merely sets certain weights to zero without changing the structure. As a result, the model's size remains the same before and after pruning, because the model remains dense with zeros replacing some of the weights. To mitigate this and reduce the amount of data transferred, sparse matrices are employed to ensure that only non-zero values are communicated to the server. Flower is optimized for transmitting model parameters as `NumPy` arrays. `NumPy` arrays are converted to sparse arrays using the `SciPy` library, which significantly reduces the data size transmitted. Initial attempts to use the `torch.Sparse` library for this conversion encountered errors, leading to the adoption of `SciPy` for reliable and efficient conversion of `NumPy` arrays to sparse matrices. This approach effectively minimizes the communication overhead in the federated learning environment.

4 Results and Evaluation

4.1 Comparison of Structured vs. Unstructured Pruning

The results indicate that global unstructured pruning consistently outperforms local structured pruning in maintaining model accuracy at a given model size. For instance:

- On the CIFAR10 dataset with ResNet12 model, unstructured pruning maintains an accuracy of 40% even with 70% pruning rate and a model size of 62.66 MB, while structured pruning, as seen in figure 12 shows no recovery in accuracy after 50% pruning with multiple rounds.
- On the MNIST dataset with ResNet12, unstructured pruning retains a high accuracy of 97% even when the model is pruned to 80%. Structured pruning, however, shows a noticeable drop in accuracy as the pruning rate increases, particularly at a 30% pruning rate where accuracy drops to 78% with a model size of 109.48 MB. Figures 4 and 6 compare structured and unstructured pruning techniques, showing accuracy versus rounds for various pruning rates. In unstructured pruning, the model's accuracy nears 95% at most pruning rates, except at 90% pruning. In contrast, structured pruning consistently shows decreasing accuracy with higher pruning rates, never recovering the original performance and highlighting a notable drop in effectiveness.
- Figure 6, 10, 12 illustrate the relationship between accuracy and the number of rounds for various pruning rates using the structured pruning technique. The data indicates that as the pruning rate increases, the accuracy consistently decreases. Notably, after 80% pruning, the accuracy levels off and shows minimal improvement, regardless of the number of rounds performed.

	Pruning Rate	0%	10%	20%	30%	40%	50%	60%	70%	80%	90%
CIFAR10 ResNet12 Structured	**Accuracy**	0.4	0.36	0.36	0.37	0.31	0.24	0.16	0.16	0.12	0.12
	Data Size (MB)	155.75	138.4	123.58	109.52	97.48	83.23	72.24	58.91	48.09	35.11
	Round	29	79	74	54	77	43	25	78	68	72
CIFAR10 ResNet12 Unstructured	**Accuracy**	0.38	0.38	0.4	0.4	0.39	0.4	0.4	0.4	0.39	0.36
	Data Size (MB)	155.75	142.45	129.15	115.85	102.55	89.25	75.96	62.66	49.36	36.06
	Round	49	30	52	35	36	25	27	39	58	46
MNIST ResNet12 Structured	**Accuracy**	0.97	0.98	0.87	0.78	0.64	0.57	0.44	0.36	0.23	0.11
	Data Size (MB)	155.74	141.51	126.13	109.48	94.24	77.12	61.22	56.46	41.73	32.01
	Round	29	41	25	48	38	38	49	41	29	1
MNIST ResNet12 Unstructured	**Accuracy**	0.97	0.97	0.97	0.97	0.96	0.97	0.96	0.97	0.97	0.49
	Data Size (MB)	155.74	142.44	129.14	115.84	102.55	89.25	75.95	62.65	49.35	36.06
	Round	15	14	10	15	15	15	15	15	15	14
MNIST Simple CNN Structured	**Accuracy**	0.92	0.93	0.82	0.73	0.61	0.56	0.42	0.34	0.16	0.11
	Data Size (MB)	0.35	0.31	0.28	0.25	0.21	0.18	0.15	0.11	0.08	0.05
	Round	20	24	24	25	22	23	23	24	14	1
MNIST Simple CNN Unstructured	**Accuracy**	0.93	0.93	0.93	0.93	0.93	0.93	0.91	0.9	0.87	0.32
	Data Size (MB)	0.35	0.32	0.28	0.25	0.21	0.18	0.15	0.11	0.08	0.05
	Round	22	20	22	25	23	23	21	25	25	25

Table 1: Experimental data for varying pruning rates, including key metrics: the highest centralized model accuracy achieved, the average model parameter size in megabytes (MB), and the number of training rounds required to reach the highest accuracy.

These results suggest that global unstructured pruning is more effective in compressing the model without significant loss in accuracy, especially when considering models trained over multiple rounds.

4.2 Impact of Pruning on Model Size and Rounds

The experiments also demonstrate that achieving better accuracy can be accomplished with a smaller model size through multiple rounds of federated learning. For example:

- On the CIFAR10 dataset with unstructured pruning, the baseline model accuracy of 38% was increased to 40% with a 70% pruned model of size 62.66 MB after only 39 rounds compared to the 49 rounds required by the baseline model to reach that accuracy. Pruning the model to 90% results in an accuracy drop of 2% but the model size is reduced from 155.75MB to 36.06MB. Figure 14 illustrates the trend of accuracy recovery across different pruning rates as the number of rounds increases.
- On the MNIST dataset using the SimpleCNN model [See figure 8], unstructured pruning sustained an accuracy of 0.93 even with a 50% pruning of model which brings the model size from 0.35MB to 0.18MB with only 1 extra round.
- The MNIST dataset using ResNet12 model shows a similar pattern where 80% pruned model achieves a 97% accuracy in same number of rounds as the baseline model with the model size reduced to 49.35MB compared to the baseline model size of 155.74MB

4.3 Large Model is Not Always Necessary

The findings demonstrate that selecting an appropriately sized model is crucial for optimal deployment for a specific deep learning task. Figure 5 presents the number of rounds required to achieve a target accuracy of 95% and the amount of data transmitted at various pruning rates for the MNIST ResNet12 unstructured pruning experiment. The data shows that while the number of bytes transferred decreases as the pruning rate increases, the number of rounds required does not consistently increase. Specifically, an unpruned model and a model pruned at 20% both require 5 rounds to reach 95% accuracy. In contrast, a model pruned at 80% only needs 3 additional rounds to achieve the target accuracy. However, the model pruned at 90% never recovers its original accuracy, suggesting that it becomes too simplified to perform effectively. A similar trend is observed in Figures 9 and 15 for the MNIST dataset with the SimpleCNN model and the CIFAR-10 dataset with the ResNet12 model. In these cases, reducing the model size through pruning does not lead to a significant increase in the number of rounds required to reach the target accuracy. This suggests that the original model size was larger than necessary for the dataset, and reducing the model size after pruning results in similar accuracy with only a minimal increase in rounds.

Conversely, Figures 7 and 11 illustrates that for the structured pruning experiment, the model fails to achieve the target accuracy once pruning exceeds 10%. This indicates a more pronounced degradation in performance with increased pruning for structured pruning compared to unstructured pruning. The models pruned with unstructured methods achieved similar or better accuracy with fewer rounds and smaller sizes compared to the full-sized model. This underscores the importance of pruning techniques in optimizing model performance while minimizing computational costs.

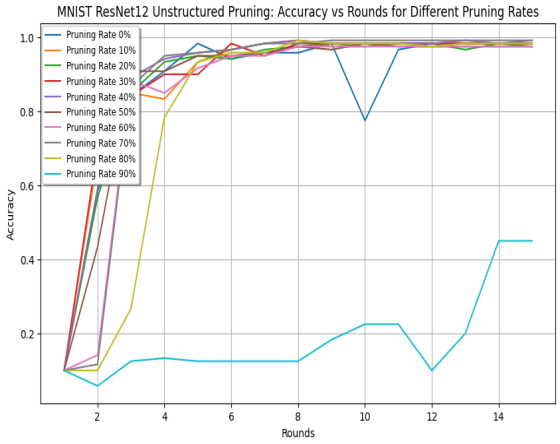

Figure 4: Accuracy vs. Rounds at Different Pruning Rates

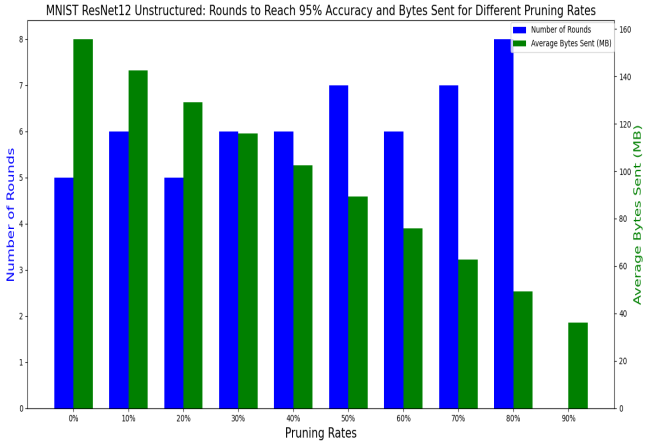

Figure 5: Rounds to 95% Accuracy and Bytes Sent at Different Pruning rates

MNIST ResNet12 Unstructured

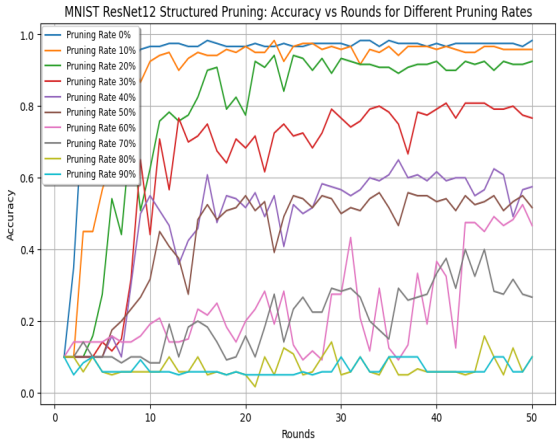

Figure 6: Accuracy vs. Rounds at Different Pruning Rates

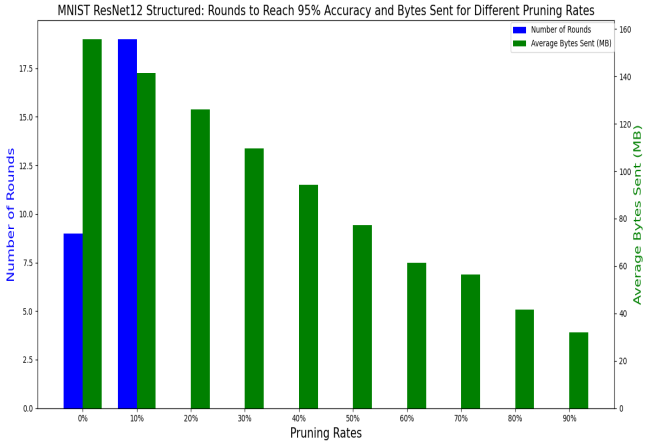

Figure 7: Rounds to 95% Accuracy and Bytes Sent at Different Pruning rates

MNIST ResNet12 Structured

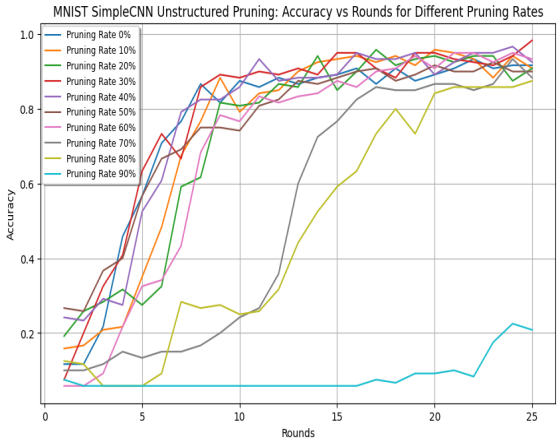

Figure 8: Accuracy vs. Rounds at Different Pruning Rates

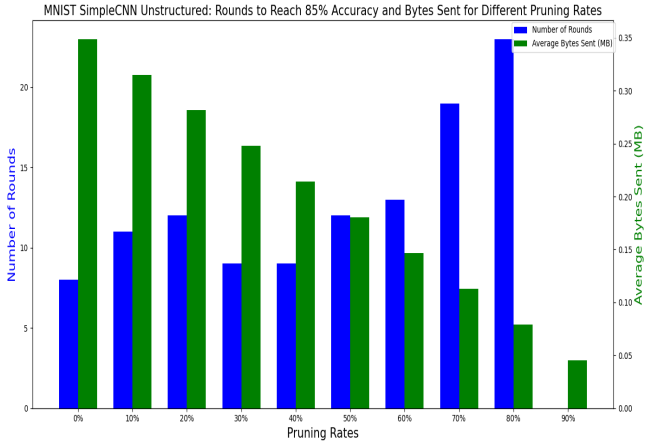

Figure 9: Rounds to 85% Accuracy and Bytes Sent at Different Pruning rates

MNIST SimpleCNN Unstructured

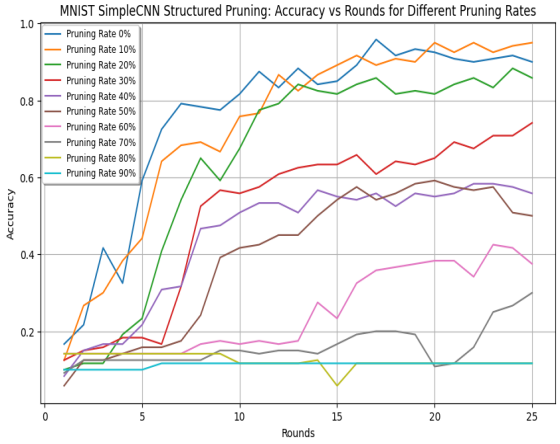

Figure 10: Accuracy vs. Rounds at Different Pruning Rates

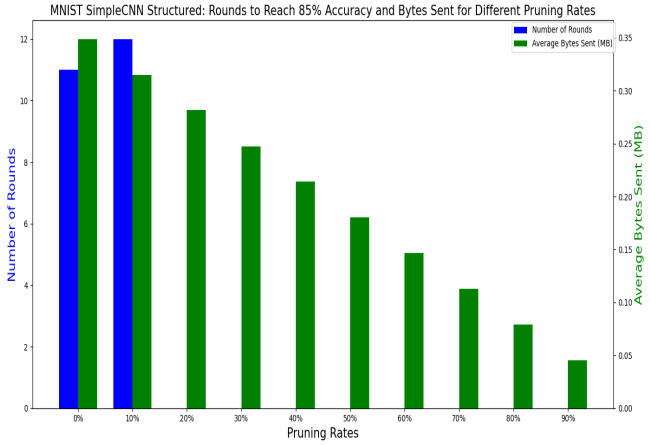

Figure 11: Rounds to 85% Accuracy and Bytes Sent at Different Pruning rates

MNIST SimpleCNN Structured

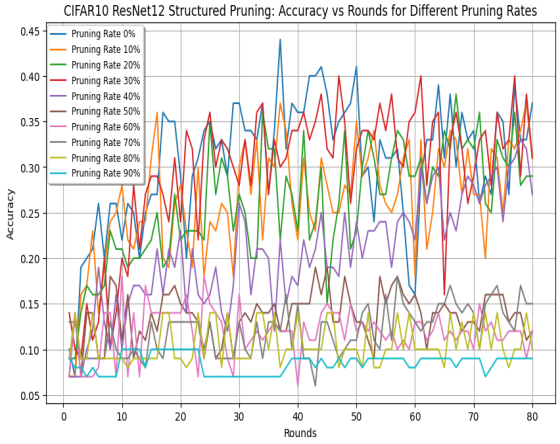

Figure 12: Accuracy vs. Rounds at Different Pruning Rates

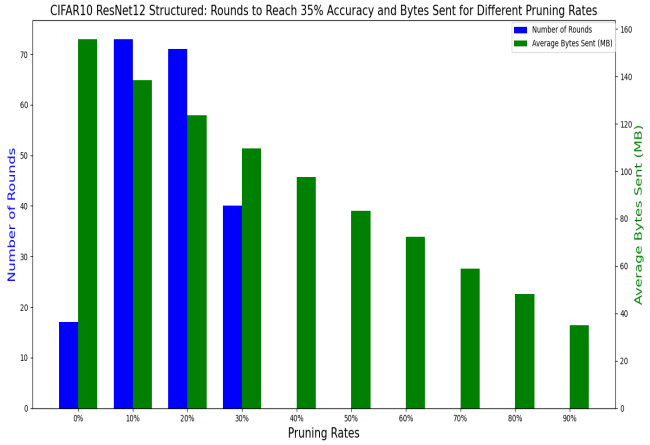

Figure 13: Rounds to 35% Accuracy and Bytes Sent at Different Pruning rates

CIFAR10 ResNet12 Structured

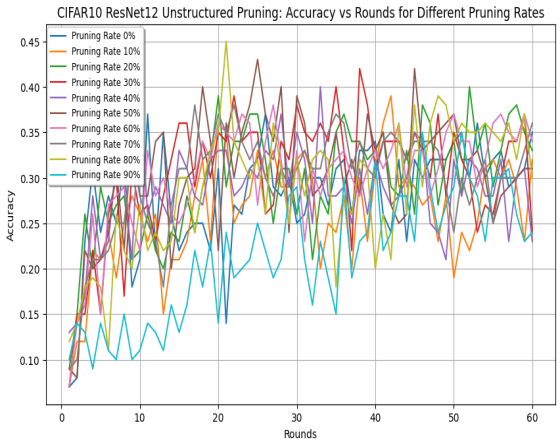

Figure 14: Accuracy vs. Rounds at Different Pruning Rates

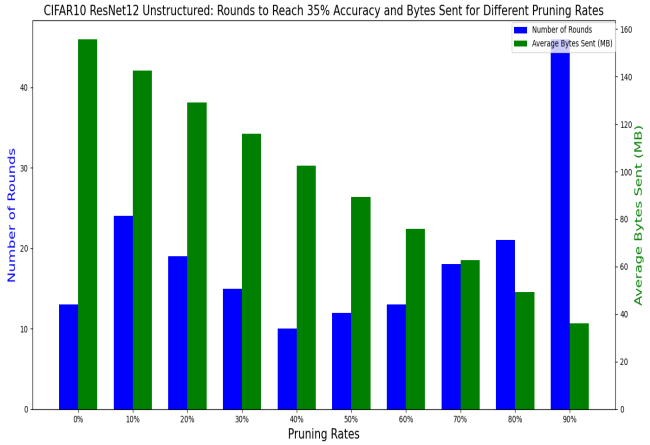

Figure 15: Rounds to 35% Accuracy and Bytes Sent at Different Pruning rates

CIFAR10 ResNet12 Unstructured

5 Conclusion

This research has explored the impact of structured and unstructured pruning as model compression strategies within the context of federated learning, particularly for scenarios involving bandwidth-constrained edge devices. Pruning is implemented at the client side using PyTorch pruning library after the model has been trained. The pruned model is further converted to a sparse matrix to reduce the number of bytes required for communication. Through extensive simulations using the Flower framework, our study demonstrates that global unstructured pruning is notably effective in preserving model accuracy at reduced model sizes compared to local structured pruning. For instance, on the MNIST dataset with ResNet12, unstructured pruning maintains a high accuracy of 97% with 80% pruning, while structured pruning suffers a significant accuracy drop as pruning increases.

A critical insight from our experiments is that achieving the highest possible accuracy does not necessitate deploying large, complex models. The experiments demonstrate that while pruning reduces the amount of data transmitted, it does not necessarily increase the number of rounds required to achieve target accuracy. For example, in the CIFAR-10 dataset with unstructured pruning, the model accuracy improves by 2% with a 70% pruning rate compared to the baseline model, and it achieves this accuracy in 39 rounds instead of the 49 rounds required by the baseline. Similarly, for the MNIST dataset, both the SimpleCNN and ResNet12 models show that reduced model sizes due to pruning result in similar or improved accuracy with only a minimal increase in the number of rounds. This suggests that the original model sizes were larger than necessary for the datasets. Instead, an optimal model size—achieved through careful pruning—can attain comparable or improved accuracy while significantly reducing the communication overhead. This is especially relevant in federated learning environments, where bandwidth and computational resources are often limited. The balance between model size, the number of training rounds, and accuracy emerges as a key factor in optimizing federated learning performance. Moreover, our findings suggest that further improvements in model compression and efficiency can be realized by integrating pruning with other techniques, such as quantization. This approach could potentially lead to even greater reductions in communication costs while maintaining or enhancing model accuracy, thereby addressing one of the major challenges in federated learning on edge devices.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial Intelligence and Statistics*, PMLR, 2017, pp. 1273–1282.

[2] T. Li, A. K. Sahu, M. Zaheer, M. Sanjabi, A. Talwalkar, and V. Smith, "Federated Optimization in Heterogeneous Networks," in *Proceedings of Machine Learning and Systems*, vol. 2, 2020, pp. 429–450.

[3] J. Wang, Q. Liu, H. Liang, G. Joshi, and H. V. Poor, "Tackling the Objective Inconsistency Problem in Heterogeneous Federated Optimization," in *Advances in Neural Information Processing Systems*, vol. 33, 2020, pp. 7611–7623.

[4] P. Qi, D. Chiaro, A. Guzzo, M. Ianni, G. Fortino, and F. Piccialli, "Model aggregation techniques in federated learning: A comprehensive survey," *Future Generation Computer Systems*, vol. 150, pp. 272–293, 2024.

[5] Z. You, K. Yan, J. Ye, M. Ma, and P. Wang, "Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks," presented at *Conference in Vancouver, BC, Canada*, 2019, p. Baseline models; iterative pruning; pruning algorithms; pruning methods; scaling factors; special operations; state of the art; Taylor expansions.

[6] R. Lin *et al.*, "Federated Pruning: Improving Neural Network Efficiency with Federated Learning," *arXiv preprint arXiv:2206.00153*, 2022.

[7] "Pruning Tutorial — PyTorch Tutorials 2.0.1+cu117 documentation," pytorch.org. `https://pytorch.org/tutorials/intermediate/pruning_tutorial.html` (accessed Jul. 31, 2024).

[8] D. J. Beutel *et al.*, "Flower: A Friendly Federated Learning Research Framework," *arXiv preprint arXiv:2007.14390*, 2020.

[9] "Flower Datasets 0.1.0," flower.ai. https://flower.ai/docs/datasets/ (accessed Jul. 31, 2024).

[10] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-Efficient SGD via Gradient Quantization and Encoding," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[11] J. Bernstein, Y.-X. Wang, K. Azizzadenesheli, and A. Anandkumar, "signSGD: Compressed Optimisation for Non-Convex Problems," *arXiv preprint arXiv:1802.04434*, 2018.

[12] W. Wen *et al.*, "TernGrad: Ternary Gradients to Reduce Communication in Distributed Deep Learning," *arXiv preprint arXiv:1705.07878*, 2017.

[13] M. Vono, V. Plassier, A. Durmus, A. Dieuleveut, and E. Moulines, "QLSD: Quantised Langevin Stochastic Dynamics for Bayesian Federated Learning," *arXiv preprint arXiv:2206.00153*, 2022.

[14] J. Wangni, J. Wang, J. Liu, and T. Zhang, "Gradient Sparsification for Communication-Efficient Distributed Optimization," in *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[15] F. Sattler, S. Wiedemann, K.-R. Müller, and W. Samek, "Robust and Communication-Efficient Federated Learning From Non-i.i.d. Data," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 9, pp. 3400–3413, 2020.

[16] X. Zhu, J. Wang, W. Chen, and K. Sato, "Model compression and privacy preserving framework for federated learning," *Future Generation Computer Systems*, vol. 140, pp. 376–389, 2023.

[17] H. Xu, K. Kostopoulou, A. Dutta, X. Li, A. Ntoulas, and P. Kalnis, "DeepReduce: A Sparse-tensor Communication Framework for Federated Deep Learning," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 21150–21163, 2021.

[18] Y. Jiang *et al.*, "Model pruning enables efficient federated learning on edge devices," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 2022, pp. 1–13, 2022, doi: 10.1109/TNNLS.2022.3166101.

[19] M. Lin *et al.*, "Hranks: Filter pruning using high-rank feature map," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1529–1538.

[20] H. Li, H. Samet, A. Kadav, I. Durdanovic, and H. P. Graf, "Pruning filters for efficient convnets," presented at *Conference in Toulon, France*, 2017.

[21] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 806–814.

[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[23] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[24] P. Molchanov *et al.*, "Pruning convolutional neural networks for resource-efficient inference," presented at *Conference in Toulon, France*, 2017.

[25] N. Lee, T. Ajanthan, and P. H. S. Torr, "SNIP: Single-shot network pruning based on connection sensitivity," presented at *Conference in New Orleans, LA, United States*, 2019.

[26] Y. He *et al.*, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4340–4349.

[27] S. Guo *et al.*, "DMCP: Differentiable Markov channel pruning for neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1539–1547.

[28] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems*, vol. 28, 2015.

[29] Y. He *et al.*, "Asymptotic soft filter pruning for deep convolutional neural networks," *IEEE Transactions on Cybernetics*, vol. 50, no. 8, pp. 3594–3604, 2019.

[30] S. Shen *et al.*, "Learning to prune in training via dynamic channel propagation," in *Proceedings of the 2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 939–945, IEEE.

[31] L. G. Ribeiro *et al.*, "Federated learning compression designed for lightweight communications," *arXiv preprint arXiv:2310.14693*, 2023.

[32] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," *arXiv preprint arXiv:1510.00149*, 2016.

[33] S. M. Shah and V. K. N. Lau, "Model Compression for Communication Efficient Federated Learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 9, pp. 5937–5951, 2023.

[34] Malekijoo, M. J. Fadaeieslam, H. Malekijou, M. Homayounfar, F. Alizadeh-Shabdiz, and R. Rawassizadeh, "FEDZIP: A Compression Framework for Communication-Efficient Federated Learning," *arXiv preprint arXiv:2310.14693*, 2021.

[35] M. Kim, S. Yu, S. Kim, and S.-M. Moon, "DepthFL: Depthwise Federated Learning for Heterogeneous Clients," in *Proceedings of the Eleventh International Conference on Learning Representations*, 2023. Available: https://openreview.net/forum?id=pf8RIZTMU58

[36] PyTorch, "PyTorch," Pytorch.org, 2023. `https://pytorch.org/` (accessed Jul. 31, 2024).

[37] "TensorFlow Federated," TensorFlow. `https://www.tensorflow.org/federated` (accessed Jul. 31, 2024).

[38] VisDrone, "VisDrone/VisDrone-Dataset," GitHub, Mar. 29, 2020. `https://github.com/VisDrone/VisDrone-Dataset`