A Federated Approach in Training Acoustic Models

Dimitrios Dimitriadis, Kenichi Kumatani, Robert Gmyr, Yashesh Gaur and Sefik Emre Eskimez

Microsoft, One Microsoft Way, Redmond, WA, USA

{didimit,kekumata,rgmyr,ygaur,seeskime}@microsoft.com

Abstract

In this paper, a novel platform for Acoustic Model training based on Federated Learning (FL) is described. This is the first attempt to introduce Federated Learning techniques in Speech Recognition (SR) tasks. Besides the novelty of the task, the paper describes an easily generalizable FL platform and presents the design decisions used for this task. Amongst the novel algorithms introduced is a hierarchical optimization scheme employing pairs of optimizers and an algorithm for gradient selection, leading to improvements in training time and SR performance. The gradient selection algorithm is based on weighting the gradients during the aggregation step. It effectively acts as a regularization process right before the gradient propagation. This process may address one of the FL challenges, i.e. training on vastly heterogeneous data. The experimental validation of the proposed system is based on the LibriSpeech task, presenting a speed-up of $\times 1.5$ and 6% WERR. The proposed Federated Learning system appears to outperform the golden standard of distributed training in both convergence speed and overall model performance. Further improvements have been experienced in internal tasks.

Index Terms: Acoustic Modeling, distributed training, federated learning

1. Introduction

Distributed Training (DT) has drawn much attention with the goal of scaling the model training processes. Since the training datasets become ever larger, the need for training parallelization becomes more pressing. Different techniques have been proposed over the years [1], aiming at a more efficient training pipeline, such as "Horovod" [2, 3] or Blockwise Model-Update Filtering (BMUF) [4]. These techniques are evaluated on metrics like throughput (without losing accuracy), model and/or dataset size, and GPU utilization. However, there are a few basic assumptions implied during DT, such as data and device uniformity, efficient network communication between the working nodes, etc. Apart from the communication/network constraints, the data uniformity is paramount for the successful training, ensured by repeated randomization and data shuffling.

On the other hand, new constraints in data management are now emerging. Some of these constraints are driven by the need for privacy compliance of the personal data and information [5]. Increasingly more data is stored behind inaccessible firewalls or on users' devices without the option of sharing for centralized training. To this end, the Federated Learning (FL) paradigm has been proposed, addressing the privacy concerns, while still processing such inaccessible data. This scheme aims at training ML models, e.g., deep neural networks, on data sets found in multiples of local end-nodes without exchanging any data between the "coordinator" and these working nodes. The general principle is based on training different versions of the model on local data samples while exchanging only updates for some of the model parameters, e.g., the network parameters or the corresponding gradients. An additional step of syncing these local models and updating the global model at some frequency is now required. There are different approaches for FL using either a central server, i.e. a "coordinator or orchestrator", or employing peer-to-peer learning, without using a central server – herein, the first approach is followed. As such, the server is responsible for the sampling and communication between the clients and the server, the model and the learning rate updating, etc.

Other differences between FL and DT lay in the assumptions made on the properties of the local data sets [6]. DT primarily aims at parallelizing local computing power, whereas FL focuses on training with heterogeneous data sets. Since DT focus is to train a single model on multiple nodes, a common underlying constraint is that all local data subsets need to be homogeneous, i.e. uniformly distributed and roughly about the same size. None of these constraints are necessary for FL, [7]; instead, the data sets are typically heterogeneous, and their sizes may span several orders of magnitude. The FL provides a more flexible training framework, relaxing some of the DT constraints.

To the best of our knowledge, a massively distributed and heterogeneous approach like the one presented here has not been applied before on Speech Recognition (SR) tasks – some work exists for KWS [8]. A sequence-to-sequence (seq2seq) approach is adopted for the Federated Learning platform (for the particular SR task) since the training of such all-neural models is much more straightforward than conventional SR systems, all while being easier automating this process. The seq2seq models have been gaining in popularity in SR tasks because acoustic, language, and pronunciation models of a conventional SR system can be combined into a single neural network [9]. There have been a variety of architectures for such models, including "Recurrent Neural Network Transducer" (RNN-T) [10], "Listen, Attend and Spell" (LAS) [11] and others. Herein, the later paradigm is adopted because it consistently provides the best offline results in our internal test sets. This seq2seq with attention model includes an encoder (similar to the traditional acoustic model), an attention layer, and a decoder (like the language model), more in Section 2.2.

This paper is focused on the Federated Learning platform, called here *"Federated Transfer Learning"* (FTL) platform, for the SR task. The paper contributions, besides the system description, are a hierarchical optimization and weighted model aggregation algorithms. The paper consists of the following sections: i. in Section 2 an overview of the current state-of-the-art in Federated Learning is provided. The overview description is not task-specific. ii. In Section 3 more theoretical justification and details about the SR system are described. Since this is the first application of FL in SR, theoretical extensions to this particular task are detailed. iii. Then, the experimental results are presented in Section 4, and iv. Finally, discussion and conclusions can be found in Section 5.

2. Background

2.1. Federated Learning

Federated Learning related work, e.g., [7], is mostly focused on communication efficiency, better optimization [12], and privacy aspects. Algorithms for FL are designed to handle training data with characteristics [6], such as data parallelism across a large number of nodes K, data imbalance on some of the clients, and data sparseness of local training examples. Although we are not explicitly concerned with data sparsity in this work, it is possible the data found in each of the clients is skewed towards different distributions – especially in SR applications, and the proposed algorithms address such issues. The presented work herein is mostly focused on the optimization side of FL, addressing challenges, such as data diversity and optimization.

Training statistical models using traditional distributed learning algorithms on real application data $\{x_i, y_i\}, i = 1, \ldots, N$ requires the following steps: copy the data on a centralized storage location, shuffle, uniformly distribute and then, train the models with them, similarly to [6]. On the other hand, the FL approach follows a different paradigm, requiring minimal data transfers, while being privacy compliant. The training constraints are more relaxed since the available computing nodes can be diverse or even inaccessible for periods of time. The proposed algorithms do not require sweeping of the entire data set for training but rather sampling of the available nodes at every given iteration. Additionally, the constraint of labeled data can also be alleviated.

2.2. Attention-based Sequence-to-Sequence models (seq2seq)

Attention-based sequence-to-sequence models (seq2seq) yield state-of-the-art performances for the SR task [11]. A seq2seq model is composed of 3 sub-networks: encoder, decoder, and attention. Given speech input $X = \{x_1, ..., x_T\}$, the encoder first converts it into a sequence of high-level representations H^{enc}. In these experiments, the encoder is a bLSTM [13] with LayerNorm [14].

$$H^{enc} = \{h_1^{enc}, ..., h_T^{enc}\} = \text{Encoder}(X). \tag{1}$$

The decoder acts as an acoustically-conditioned language model. For predicting a certain token y_n, the acoustic signal to be used for conditioning is summarized by the attention module. For every decoder time-step n, attention generates alignments α_n over H^{enc}, and a corresponding context vector c_n. The attention layer is a location-aware attention, as in [15].

$$c_n, \alpha_n = \text{LocationAwareAttention}(d_n, \alpha_{n-1}, H^{enc}) \tag{2}$$

Here d_n is the decoder state vector at time n. The context vector, c_n, is leveraged by the decoder LSTM as below:

$$d_n, h_n^{dec} = \text{Decoder}(y_{n-1}, c_{n-1}, h_{n-1}^{dec}), \tag{3}$$

$$y_n = \text{DecoderOut}(c_n, d_n) \tag{4}$$

where y_n the output model hypothesis.

The Decoder is a multi-layer LSTM while DecoderOut consists of an affine transform with a Softmax output layer. The model is trained with cross entropy loss $\mathscr{L}(\cdot)$ between $\mathbf{Y} = \{y_1, y_2, ..., j_i\}$ and reference labels $\mathbf{R} = \{r_1, ..., r_N, \langle eos \rangle\}$

$$\mathscr{L}(\mathbf{Y}, \mathbf{R}) = -\sum_n y_n log(r_n) \tag{5}$$

3. Proposed Approach

The developed FTL platform is a simulation of the FL training process, while ignoring both the communication and security/privacy aspects of the task. Although a seq2seq SR model [9] is used as an example, the findings and conclusions can be generalized to other tasks, as well .

3.1. System Description

The proposed system, as in Figure 1, consists of a pool of K clients with a fixed dataset per client. Contrary to DT, the training data is neither uniform nor reshuffled after every epoch – the data segregation remains fixed throughout the task. For the FTL scenario, randomly sampled $N \ll K$ clients are processed in every iteration T, and then returned to the pool, i.e. random sampling with replacement. Using just these N clients without loss in performance provides additional flexibility unique to the proposed FTL platform.

Once these N clients finish processing data, the updated models $\tilde{w}_T$ are returned to the server for aggregation, and a global gradient is estimated. This gradient is used to update the global model $w^{(s)}$ before the next iteration $T+1$. Due to this sampling of clients, the sweeping of all data takes longer. However, theoretical justification[1], and experimental results have shown that it is neither necessary for reaching an optimal point nor detrimental for the model performance.

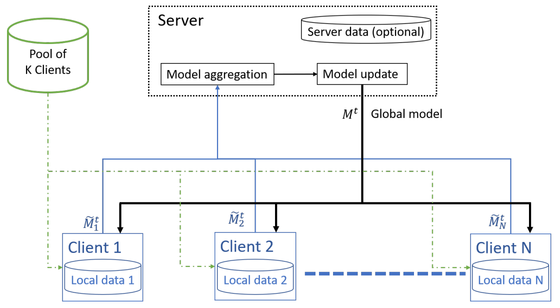

Figure 1: *FTL Platform: Processing N clients per batch out of a pool of K clients.*

The FTL simulation platform allows the faithful simulation of the FL system while excluding aspects such as encryption. Due to the large size of our production-scale SR models, we allow the simulation platform to leverage multiple GPUs so that the training time remains reasonable, on the order of hours or days instead of weeks or months. At the same time, we required the platform to allow the simulation of an arbitrary number of clients on a fixed and typically much smaller number of GPUs. To address these requirements, the simulation is implemented as an MPI program with P processes [16], each of which has access to a dedicated GPU in a (potentially multi-node) GPU cluster. The process with rank 0 simulates the server while the remaining $P-1$ processes are used as workers, simulating the model training process of the clients.

In each iteration, the server randomly samples $N \ll K$ clients to participate in the training, as described above. The simulation platform executes the model training for these N clients by dispatching training tasks to the workers: Whenever

[1]The expectation of the coverage percentage is given by $E(Y) = 1 - ((N-1)/N)^k$, where N is the population, and k is the number of iterations.

there is an idle worker, the client training code and all necessary parameters (global model, learning rate, the identifier of client data set, etc.) are sent, simulating the training process of that specific client. The worker performs the computation for that client and sends the results (locally trained model, loss, etc.) back to the server process. Next, the server updates a dictionary with the available resources and the remaining clients to be processed. This simple scheduling process continues until all N clients have been processed. To preserve the memory on the server, we aggregate the client models in a streaming fashion as they are returned by the workers. As an example in the case of simple model averaging, the server aggregates all client models into a single copy of the model that is stored in GPU memory and then divides the model parameters by N to compute the average. Once the aggregated model has been computed, the server updates the global (seed) model and re-iterates.

3.2. Hierarchical Optimization

Mini-batch optimization methods, extending classical stochastic methods to process multiple data points at a time, have emerged as a popular paradigm for FL [17]. Approaches like "Federated Averaging" (FedAvg) [6], a method based on averaging local stochastic gradient descent (SGD) updates, are proposed. FedAvg can generalize well while showing it can significantly improve performance in terms of speed-ups.

Herein, a hierarchical optimization process is proposed, further enhancing FedAvg. The training process is held on two tiers: first, on the client-side using a "local" optimizer, and then on the server with a "global" optimizer after aggregating the client gradient estimates. This two-level optimization approach combines the merits of FedAvg with additional speed-ups due to the second optimizer. Further, aggregating the gradient estimates is shown beneficial due to the inclusion of more data per iteration. The proposed algorithm appears to converge faster than Horovod, after data volume normalization [2]. Additionally, it offers more flexibility and enhanced optimization capabilities due to the fact that various combinations/pairs of optimizers can be used for the client/server setups.

In more detail, the j^{th} client update runs t iterations with $t \in [0, t_j]$, locally updating the seed model with (6) (herein, without loss of generality showing for the SDG optimizer) with a learning rate of η_j,

$$w_{t+1}^{(j)} = w_t^{(j)} - \eta_j \nabla w_t^{(j)} \tag{6}$$

where $w^{(j)}$ the local model in j^{th} client.

The j^{th} client returns a smooth approximation of the local gradient $\tilde{g}_T^{(j)}$ (after these t_J iterations) as the difference between the latest updated model with the previous global model $w_{T-1}^{(s)}$

$$\tilde{g}_T^{(j)} = w_T^{(j)} - w_{T-1}^{(s)} \tag{7}$$

The distributed training process is synchronous for these N clients. The gradients $\tilde{g}^{(j)}$ are estimated in time steps T (or equally when all clients have processed their data after t_j iterations). Note that estimating the gradients g_T is difficult and memory intensive. Herein a difference approximation $\tilde{g}_T^{(j)}$ is used instead.

After all the gradient samples are weighted and aggregated accordingly, Section 3.3, the global model $w_T^{(s)}$ is updated as in (8) (herein also using SGD),

$$w_T^{(s)} = w_{T-1}^{(s)} - \eta_s \sum_j \alpha_T^{(j)} \tilde{g}_T^{(j)} \tag{8}$$

where $\alpha_T^{(j)}$ are the weights for the aggregation step, as below.

The process described above is a simple form of "online training" [18]. While updating, the seed model can be drifted away from the original task. In order to ensure compatibility with previous tasks without increasing the network capacity, we propose an additional training iteration over held-out data on the server side (9), after the model aggregation and update step. This way, the model can be "steered" back in a direction matching the held-out data. A "gentle" update of the model can avoid diverging too much from the task of interest. This is particularly useful for the case of imbalanced and/or vastly heterogeneous data.

$$w_{\tilde{t}+1}^{(s)} = w_{\tilde{t}}^{(s)} - \eta_w \nabla w_{\tilde{t}} \tag{9}$$

The convergence speed for the hierarchical optimization scheme is improved by a factor of $\times 1.5$, without any impact on performance. Also, the communication overhead is significantly lower since the models are transferred twice per client and iteration (instead of transmitting the model gradients after every mini-batch, as in [19]).

3.3. Weighted Model Averaging

When dealing with heterogeneous data, there are a few challenges for the aggregation step, (8). First, not all clients contain data that are represented by the model; consequently, the corresponding training loss is expected significantly higher. In such cases, the model moves to a direction that is mostly different from the rest of the models. Also, the quality of a particular data partition (local client data) might be different, leading to noisier gradients. Either way, these gradients should be processed differently than the rest. The proposed solution here is to use weights during the aggregation step, i.e. of the local gradients $\tilde{g}_T^{(j)}$, in (7). This approach is to use the training losses as weighting coefficients. Although weighted aggregation does not significantly affect the overall WER performance (at least on the LibriSpeech task, where data is more homogeneous), it makes the training convergence significantly faster. The weighting process can be seen as a type of regularization de-emphasizing gradient directions, where the models diverge too much. Thus, the back-propagation updates are based on the less noisy mini-batch gradients.

This "deterministic" approach utilizes the negative training loss coefficients $\mathscr{L}$, in Eq. (5), as weights $\alpha_t^{(j)}$, after passing them through a $Softmax(\cdot)$ layer, $\alpha_t^{(j)} = \exp(\mathscr{L}_t^{(j)}) / \sum_i \exp(-\mathscr{L}_t^{(i)})$. Higher values for $\mathscr{L}^{(j)}$ coefficients can be seen as an indication of batches that are not well represented by the model. Possible sources of such loss values are either data of bad quality, e.g., noisy data, or data distributions further apart from the model. Either way, the resulting model (after the training step) will have been "moved" more, further apart from the rest of the models; thus, the aggregation will be noisier. Batches[3] where the training losses are of similar magnitude would be expected to move the model in a similar direction; thus, the aggregation process will be better aligned. Such alignment of the aggregated gradients is also beneficial for

[2] The comparison between Horovod and the proposed algorithm are with similar amounts of data processed.

[3] We interchangeably use 'batch' and 'client' in this analysis – each client can be seen as a large batch of data, while FL training.

the convergence speed, as shown in the experimental part. Since the Softmax layer considers all the available training steps, it de-emphasizes those with larger values of the corresponding losses.

4. Experiments and Results

The LibriSpeech task [20] (LS task)is used as the experimental testbed. The dataset contains about $1k$ hours of speech from $2.5k$ speakers reading books. In this experiment, a 6-layer bLSTM, with dropouts, is used for the encoder, 2 layers of uni-directional LSTM are used for the decoder, and finally, a conventional location-aware content-based attention layer with a single head is used. The input features are 80-dim log mel filter-bank energies, extracted every 10 *msec*. 3 frames of features are stacked and applied on the encoder, $16k$ subwords based on a unigram language model are used as the recognition unit. Initially, a state-of-the-art seq2seq model is trained using Horovod on the entire training set of $920h$. The performance of this model provides the lower bound of the WER since all the data is used in a centralized manner.

For the FL-related experiments, the training is split into two parts of 460h each, with no overlapping speakers. The first part is used to train a seed model, without ever using this data again. Then, the 2nd part of the dataset is used to simulate online training, under the FL conditions. We will follow two different directions for the FL training process: first, the training set is split into 7 distinct parts without reshuffling the data again (contrary to Horovod or other DT approaches). The splits are random, with no overlapping speakers across them. The second direction is to split according to the remaining 1100 speakers. Each one, either 7 or 1.1k, of the partitions is assigned to a client node. In the FTL framework, the clients are unaware of the rest of them – only the server knows which the clients are used, randomly sampling which of them will be aggregated. The number of sampled clients N in our experiments varied from 25 to 400, higher N being better but with minor fluctuations in overall performance. Based on the compromise between communication overheads and memory usage, we henceforth set $N = 100$.

The weighting approach for α_T^j is based on the training loss, herein noted as $Softmax$-weighting.

LibriSpeech Clean Task		
Training Scenario		WER (%)
Centralized	SotA (lower bound)	4.00%
	training on 1st 50% of LS(seed)	5.66%
	online training on 2nd 50% of LS	4.61%
FTL	FL-based Single Optimizer	4.55%
	Hier. Optim. (7 clients)	4.51%
	Hier. Optim. (1.1k clients)	4.45%
	+ Softmax Weighting	4.41%

Table 1: *System evaluation on the LibriSpeech clean test set, training an offline seq2seq with attention model.*

The first 3 rows in Table 1 are with centralized training (Horovod), with the lower bound in performance coming from the model trained on the entire dataset. The 4% WER for this model appears inline with the literature. The 2nd model ("online training" row) is based on the seed model initially trained on the 1st half of the data till convergence. The model is online trained with the 2nd half of data. In both steps, Horovod is used.

Different strategies for model aggregation were investigated, such as FedAVg, i.e. "single optimizer" row in the Table, or hierarchical optimization using optimizers such as Adam, LAMB, LAR, and SGD. Combinations of the server/client optimizers were also investigated. The differences in performance for these combinations of optimizers were rather limited, and for the sake of space, are not listed here. However, a state-less optimizer on the client-side is adopted as the standard, because the initial model is changing after each iteration[4] and therefore, keeping the state of the previous iteration/model as part of the local optimizer did not make sense. The combination used for all the experiments is Adam/SGD for the server/client.

The next experiment was to transition to a per-speaker partitioning of the data, i.e. creating 1100 partitions and an equal number of clients. As mentioned above, 100 clients per iteration are sampled out of the pool of 1100, processed and finally aggregated. The transition from a homogeneous data split, such as the case of 7 partitions, to a more heterogeneous per-speaker partition lightly improved the performance of the model. This can be explained by the additional diversity provided when aggregating such client models. However, due to this diversity, the convergence during training required additional iterations.

The proposed weighted aggregation addresses this issue by de-emphasizing the gradients from batches (or equally clients) poorly modeled. Although the overall performance was not impacted (particularly in the case of the LS task), the overall convergence speed was improved by a factor of around $\times 1.5$. Approximately 650 iterations are required for the case of unweighted aggregation versus only 440 for the Softmax-based weighted average. The variations in performance between different approaches are limited; however, the task is quite homogeneous. Further improvements in other in-house tasks, e.g., adaptation on presentation sessions, have been realized, as well.

5. Discussion and Future Work

Herein, a novel Federated Learning platform for Speech Recognition tasks is presented. This is the first of its kind as far as the authors know. Herein, Federated Learning approaches for other tasks were investigated and compared with the proposed ones. Although the discussion about the platform is focused on the task in hand, this platform can be easily generalized to other tasks. Currently, we are working on other classification tasks using the FTL platform, employing other modalities.

In addition to those approaches, we are presenting novel algorithms addressing challenges unique to the Speech Recognition scenario. This novel approach of weighting the gradients between mini-batches allows for enhanced convergence speed-ups and improved model performance. The proposed gradient aggregation scheme acts as a regularizer de-emphasizing batches where the data are not well modeled. Herein, a weighted gradient aggregation algorithm is described enabling $\times 1.5$ speed-up and 6% WERR on LibriSpeech task.

6. Acknowledgements

We would like to thank Masaki Itagaki, Michael Zeng, Xuedong Huang, Veljko Miljanic and Frank Seide for their project support and technical discussions. Special mention to Ray Zhou, Lei Ma and team for building the FTL SDK including encryption and the other FL features.

[4]As discussed in Section 3, the server aggregates the client models and updates the seed (server) model. Then, this model is re-iterated to the clients.

7. References

[1] T. Ben-Nun and T. Hoefler, "Demystifying Parallel and Distributed Deep Learning: An In-depth Concurrency Analysis," *ACM Computing Surveys*, , no. 65, 2019.

[2] Alexander Sergeev and Mike D. Bals, "Horovod: Fast and Easy Distributed Deep Learning in TensorFlow," *arXiv preprint arXiv:1802.05799*, 2018.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D.G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A System for Large-Scale Machine Learning," *arXiv preprint arXiv:1605.08695*, 2016.

[4] K. Chen and Q. Huo, "Scalable Training of Deep Learning Machines by Incremental Block Training with Intra-block Parallel Optimization and Blockwise Model-update Filtering," in *Proc. ICASSP*, 2016.

[5] Ben Wolford, "A Guide to GDPR Data Privacy Requirements," `https://gdpr.eu/data-privacy/`.

[6] J. Konecny, B.H. McMahan, and D. Ramage, "Federated Optimization: Distributed Optimization Beyond the Datacenter," *arXiv preprint arXiv:1511.03575v1*, 2015.

[7] Q. Li, Z. Wen, Z. Wu, S. Hu, N. Wang, and B He, "A Survey on Federated Learning Systems: Vision, Hype and Reality for Data Privacy and Protection," *arXiv preprint arXiv:1907.09693v4*, 2020.

[8] D. Leroy, A. Coucke, T. Lavril, Gisselbrecht T., and J. Dureau, "Federated Learning for Keyword Spotting," *arXiv preprint arXiv:1810.05512v4*, 2019.

[9] Chung-Cheng Chiu et al, "State-of-the-art Speech Recognition With Sequence-to-Sequence Models," *arXiv preprint arXiv:1712.01769*, 2018.

[10] A Graves, "Sequence Transduction with Recurrent Neural Networks," *arXiv preprint arXiv:1211.3711*, 2012.

[11] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, Attend and Spell," *arXiv preprint arXiv:1508.01211*, 2015.

[12] O. Shamir, N. Srebro, and T. Zhang, "Communication Efficient Distributed Optimization Using an Approximate Newton-type Method," *arXiv preprint arXiv:1312.7853*, 2013.

[13] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition," *Artificial Neural Networks: Formal Models and Their Applications*, 2015.

[14] J.L. Ba, J.R. Kiros, and G. E. Hinton, "Layer Normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[15] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-Based Models for Speech Recognition," *arXiv preprint arXiv:1506.07503*, 2015.

[16] Message Passing Interface Forum, "Message-Passing Interface (MPI) standard, version 3.0," `https://www.mpi-forum.org/docs/mpi-3.0/mpi30-report.pdf`.

[17] T. Li, A. K. Sahu, A. Talwalkar, and V Smith, "Federated Learning: Challenges, Methods, and Future Directions," *arXiv preprint arXiv:1908.07873v1*, 2019.

[18] D. Sahoo, Q. Pham, J. Lu, and S. Ho, "Online Deep Learning: Learning Deep Neural Networks on the Fly," *arXiv preprint arXiv:1711.03705*, 2017.

[19] H.B. McMahan, E. Moore, D. Ramage, S. Hampson, and B.A.y. Arcas, "Communication-efficient Learning of Deep Networks from Decentralized Data," in *Proc. International Conference on Artificial Intelligence and Statistics*, 2017, pp. 1273—1282.

[20] V. Panayotov, G. Chen, Daniel Povey, and S. Khudanpur, "LibriSpeech: an ASR corpus based on public domain audio books," in *Proc. International Conference on Acoustics, Speech and Signal Processing*, 2015.