VersatileFL: Volatility-Resilient Federated Learning in Wireless Edge Networks

JinYi Yoon*, Jeewoon Kim†, Yeongsin Byeon‡, and HyungJune Lee*
*Department of Computer Science and Engineering, Ewha Womans University, Seoul, South Korea
†Department of Computer Science, University of California, Irvine, CA, USA
‡School of Computing, KAIST, Daejeon, South Korea
yjin3012@ewhain.net, jeewoonk@uci.edu, yeongsinbyeon@kaist.ac.kr, hyungjune.lee@ewha.ac.kr

***Abstract*—In the era of artificial intelligence (AI), deep neural networks (DNNs) become larger using a massive amount of data, and thus, they are trained via cooperative computing devices (e.g., GPUs or servers) based on federated learning. As computation and data generation move to the edge due to privacy, latency, or bandwidth issue, DNN with edge devices has been investigated. However, edge devices are wirelessly connected and mostly incur fragile connectivity. We propose *VersatileFL*, a novel volatility-resilient deep learning framework under hostile environments. We address short-term and long-term volatility: 1) versatile distributed learning against short-term fluctuation by substituting the missing intermediate values with the past or approximated values and 2) model rearrangement with runtime connectivity diagnosis against long-term variation by adaptively adjusting the partitioned model for the impaired. We have demonstrated that *VersatileFL* has achieved 62.0 % and 31.9 % higher performance than hostile learning without a maintenance scheme against the short-term and long-term volatility, respectively.**

***Index Terms*—Federated Learning, Distributed Learning, Split Learning, Model Parallelism, Wireless Networks, Network Volatility, Edge Intelligence**

I. Introduction

With the advance of Artificial Intelligence (AI), Deep Neural Networks (DNNs) have been extended with numerous parameters and deep layers to extract meaningful information from a vast amount of data with complicated tasks. However, in the era of the Internet of Things (IoT), data is mostly and constantly generated from the user-side. It is inefficient to upload all of the raw data into servers to process data-driven tasks, usually incurring some non-trivial problems such as lack of bandwidth, data privacy, or network latency [1]. Therefore, the existing (semi-) centralized AI frameworks may not be desirable for such applications. To tackle the problem, *federated learning* towards the edge have been proposed, where tasks and data sources are mostly requested at the user-side [2]. Different from its own purpose, most works mostly focus on the inference at edge devices, whereas the actual training is still being processed at high-computing servers [3]. Considering lightweight yet efficient learning and inference at the edge side has been a key challenge for edge intelligence.

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2021R1A2B5B01002906), and by the Institute of Information & communications Technology Planning and Evaluation (IITP) grant funded by the Korea government (MSIT) (No. RS-2022-00155966, Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University)). The corresponding author is HyungJune Lee.

As a critical issue, edge devices are wirelessly connected with each other, often leading to transmission link failure or device malfunctioning. Most of previous works on federated learning under wireless links focus on how to model a neural network to learn intrinsic patterns well, improve the inference accuracy, enable fast learning, or build a stubborn communication technique [4]–[6]. It has implicitly been assumed that learning paths are almost flawless during propagation, without considering possible imperfect intermediate outputs [7]. However, well-performing models can no longer work in case of missing parameters, omitted pass, misleading data, or fallacy in computation, incurring the learning collapse. The real-world volatile wireless environment often leads to substantial link-wise or device-wise failures.

In this paper, we propose a versatile model partitioning strategy that addresses both short-term and long-term network fluctuations in a network of edge devices. Due to the inherent volatility in wireless networks, device connectivity changes unexpectedly. By introducing a "versatile" distributed learning called *VersatileFL* allows a model to be trained under the imperfect delivery caused by link-wise failures. Moreover, since wireless networks vary over time, some long-term link connectivity changes, device malfunctioning, or available resource inflow can often occur more severely. For run-time rearrangement, we periodically analyze network dynamics and discover a sub-model rearrangement scheme by monitoring the run-time network connectivity.

To the best of our knowledge, *VersatileFL* is the first work to propose a volatility-resilient distributed learning for edge networks against both short-term and long-term variations. We have demonstrated that the proposed scheme outperforms edge learning without management with 62.0 %, while incurring slightly higher transmission overhead with a factor of 3.07. Our contributions can be summarized as follows:

- We propose a novel distributed learning mechanism, *VersatileFL*, which can compensate for a volatile link or device variations in a network of edge devices.
- By incarnating necessary tasks upon short-term or long-term connectivity volatility, we provide a way to make edge-level training stable yet efficient.

- We validated *VersatileFL* with extensive experiments under various network dynamics. Our approach has outperformed other counterpart algorithms and demonstrated the efficacy of our key claims.

II. Related Work

Our work applies volatile federated learning on top of the model parallelism in a distributed fashion.

The model parallelism assumes that the delivery among workers is stable via a wired connection such as multiple GPUs or servers [7]. However, in the edge network context where wireless communication is a norm, the overall overhead highly depends on how to divide the network. It is directly correlated to transmission cost and intermediate failures. For example, if a device takes charge of the whole neurons from a single layer (via *vertical* partitioning), it would cost less communication among devices, but would instead be risky to lose the whole layer-to-layer propagation with only a single failed pass case [8]. On the other hand, if a device is responsible for partial neurons from each layer across the whole layers (via *horizontal* partitioning), it would be more relatively easy and stable to secure a complete propagation pass across the layers, but excessive communication among devices that take charge of the neurons at the same layer is required to make progress towards a successful learning pass.

Recent works solve the problem of resilient distributed learning over wireless edge devices in the area of federated learning, distributed inference, or multi-agent reinforcement learning [4]–[6]. They mainly solve the problem of model design by optimizing resource consumption, latency, communication reliability, or overhead [9]. More closely related to this problem, *EdgePipe* [10] presents a resilient distributed learning structure with learning acceleration, but any extra backup scheme for edge-based learning under volatility among devices has not been well investigated. However, previous works on resilient learning in wireless networks can be fragile in case of dynamic failures of both links and nodes. As a subsidiary framework to supplement the volatility, some backup workers are deployed in distributed learning [11]. Although they can be a solution for fast and reliable learning in place of a parameter server, a more in-depth approach that can address both intermittent and permanent network volatility needs to be devised in the inherently dynamic wireless networks.

III. System Model

We consider the problem of deep learning in a distributed edge network consisting of wireless devices. To reduce the computational burden at the edge, we take a model parallelism-driven approach. Model partitioning makes even a large training model feasible at the low-end devices. We propose *VersatileFL*, a novel collaborative learning framework over DNNs, where neurons in the adjacent layers are fully connected. As in Fig. 1, it is supposed that a DNN is partitioned into multiple edge devices. The devices in charge of the input layer begin a round-trip with on-board training data. Each device takes charge of its own sub-model shard, and the training is processed through wireless links in the order of layers.

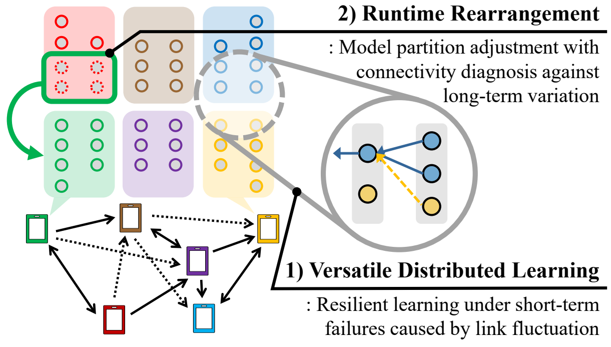

2) Runtime Rearrangement
: Model partition adjustment with connectivity diagnosis against long-term variation
1) Versatile Distributed Learning
: Resilient learning under short-term failures caused by link fluctuation

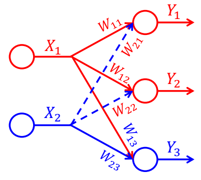
Fig. 1. Overview of *VersatileFL*, a volatility-resilient learning architecture

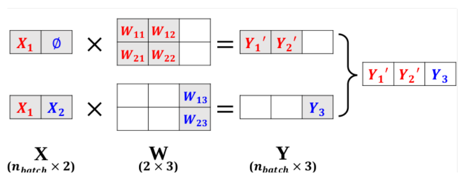
It is assumed that edge devices communicate with each other via wireless radio such as IEEE 802.11, without any help of a cloud service. Due to the inherent fluctuations, links between edge devices can be disconnected intermittently. As another critical issue, if some devices malfunction, the corresponding partial sub-models may completely be lost during training. Our goal is to make a training part durable even under volatility by diagnosing and managing the delivery status among the partitioned networks.

To make a distributed learning resilient under volatility, we suggest two ways to derive a sturdy and agile mechanisms by taking into account the short-term and the long-term volatility.

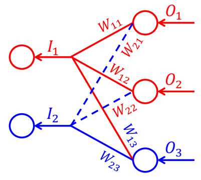
1) Versatile Learning against Short-Term Volatility: In wireless networks, we cannot always guarantee seamless delivery among edge devices. Intermittent communication failures occur unexpectedly since wireless communication is fragile due to the dynamics of the wireless environment. Even though there are many supplementary schemes including retransmission under volatility, it mostly incurs excessive transmission overhead, which critically affects performance at the edge side. Without any additional support from transmission recovery, it would be interesting and necessary to let learning itself be proceeded with the best effort even with imperfect propagation. We present a versatile learning scheme, which sub-optimally approximates the outputs in the forward and the backward passes by substituting missing values. This approach enables robust training smoothly using intermediate pass patches. More details are described in Sec. IV.

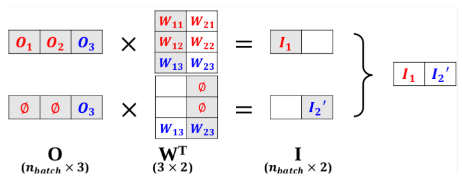
2) Model Rearrangement against Long-Term Volatility: In a relatively stable architecture using link fluctuation maintenance, some long-term dynamics can destroy the learning even more severely, such as device malfunctioning or obstruction between devices. Also, the current network configuration accordingly becomes different from the initial network connection. Due to these kinds of long-term variations, some more structural arrangement is necessary. We propose a cost-effective runtime maneuver, which detects more long-term network change or loss and in-flow of devices, and appropriately rearranges the model allocation. Without any additional transmission overhead, we can detect both link and node failure cases by logging reception history, and then optimize a deep learning structure to make learning stable. The detailed process is described in Sec. V.


(a) Forward passes

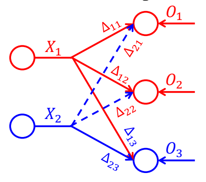


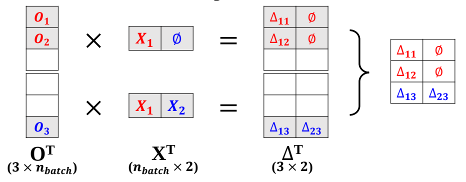
(b) Matrix multiplication in forward


(c) Backward passes



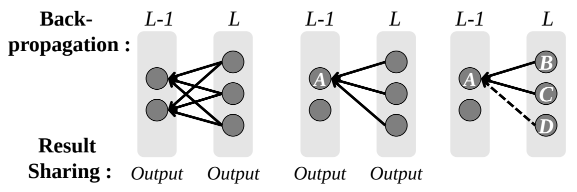
(d) Matrix multiplication in backward


(e) Gradient computation



(f) Matrix multiplication in gradient computation

Fig. 2. The forward, backward propagation, and gradient computation, where *Dev. 1* and *Dev. 2* take charge of red and blue, respectively. The solid line denotes a successful connection, with the dotted line for a failed connection.



(a) Layer-wise (b) Neuron-wise (c) Link-wise

Fig. 3. Granularity basis for the saved gradients for backpropagation and result sharing in *VersatileFL*, where backpropagation passes from layer L to $L-1$, and result sharing occurs within an output layer

IV. Versatile Distributed Learning against Short-Term Volatility

We present a versatile distributed learning mechanism to enable learning under fragile wireless propagation. Instead of directly using uncertain intermediate computed values at the forward or the backward pass, which may be caused by instantaneous link fluctuations, we accept the innate volatility during the training and control whether a newly computed value can be reflected or not. If the new update is not ready to be reflected, we decide to apply past or approximated values. Further, we suggest a dynamic update mechanism during each pass for more stable training progress.

We devise to partition the *dense* layers to make deep learning viable on edge devices, which account for the most computation loads in DNNs [8]. We focus on the dense layers, which are mainly partitioned in model parallelism, and other types of layers such as convolutional layers or pooling layers to improve the learning performance can be divided by filters or easily stacked as an additional layer regarding the computation and communication overhead.

A. Forward Propagation

During a forward propagation, a training batch is injected into the neural network with certain parameter values. In case that some specific forwarding paths become unstable toward a successful propagation to the output layer, it may be harmful to use the intact intermediate values without any careful investigation.

$$Y_j = \Sigma_{i=1}^{N_{l-1}}(X_i \cdot W_{i,j}) \tag{1}$$

where Y_j and $W_{i,j}$ are the outputs and the weights, respectively, with N_l neurons at layer l. X_i is the input of the layer, which is the output of the previous layer, and X_i is zero, when transmission from neuron i to j, i.e., $TX(i \rightarrow j)$, fails. For example of the forward propagation in Fig. 2(a), $Y_1 = \Sigma_{i=1}^{2}(X_i \cdot W_{i,1}) = X_1 \cdot W_{1,1} + X_2 \cdot W_{2,1}$, but when $TX(1 \rightarrow 2)$ succeeds, but $TX(2 \rightarrow 1)$ fails, *Dev. 1* computes Y_1 and Y_2 by substituting X_2 by zero (as one of the substitute values) as $Y_1' = X_1 \cdot W_{1,1} + 0 \cdot W_{2,1} = X_1 \cdot W_{1,1}$ in Fig. 2(b).

A successful forward propagation until the last output layer is very crucial for its subsequent backward propagation. Only after calculating the correct loss value at the output layer, can the backpropagation continue to compute the gradients and update the weight parameters. For example, if only half of the activation at one layer is successfully delivered to its next layer in DNN with four layers, only $0.5^{4-1} = 0.125 = 12.5$ % of training is valid; the remaining 87.5% misleads the training.

To prevent learning from being dominated by intermittent imperfect propagation, we introduce a forward criterion, $thres_{fw}$, to filter out problematic forward passes. If all of the layer-to-layer success rates are larger than $thres_{fw}$, a forward pass is allowed to continue. For example, if $thres_{fw}$ is set to 0.5, the forward success rate in the sequence of [0.75, 1, 0.5] with four layers is regarded as valid, and the backward passes are proceeded with the outputs. However, in case of [0.75, 0.25, 0.5], some rates (i.e., 0.25 and 0.5) do not meet the criterion, and the case is considered as invalid.

For a training batch that satisfies the forward criterion up to the output layer, we use the batch to calculate the loss and continue with its subsequent backward propagation.

B. Backward Propagation

For backpropagation, it is important to determine which value to be passed to the precedent layers, and also how to compute the gradient values of the loss in case of disconnection. Similar to the forward pass, the backpropagation error can be computed as follows:

$$I_i = \Sigma_{j=1}^{N_l}(O_j \cdot W_{i,j}) \tag{2}$$

where O_j is zero when $TX(j \rightarrow i)$ fails. As in Fig. 2(c), $I_2 = \Sigma_{j=1}^{3}(O_j \cdot W_{2,j}) = O_1 \cdot W_{2,1} + O_2 \cdot W_{2,2} + O_3 \cdot W_{2,3}$, but if $TX(1 \rightarrow 2)$ fails, O_1 and O_2 are regarded as zeros. Thus, as in Fig. 2(d), we approximate the value as $I_2' = 0 \cdot W_{2,1} + 0 \cdot W_{2,2} + O_3 \cdot W_{2,3} = O_3 \cdot W_{2,3}$.

To compensate for the missing values due to disconnection, *VersatileFL* saves past gradients that have successfully been calculated with a previous batch. Then, it updates the weights with the previously saved gradients when the current gradients are missing due to disconnection.

Although loss is more relevant to representing each learning step, we decide to save and then reuse the gradients due to the time difference of forwarding outputs and backwarding loss. The gradients using the Gradient Descent Optimizer [12] are calculated as follows:

$$\Delta_{i,j} = X_i \cdot O_j \tag{3}$$

If we save the loss instead of gradients at a certain time t, the saved value of O_j at t is used with another forwarding value of X_i. This time gap between the input and the saved loss consequently leads to inaccurate direction, and thus, saving the gradient can make it more adequate. We can compute the gradient with the forward value X_i and the backward loss O_j, as illustrated in Figs. 2(e) and 2(f).

According to the granularity of the saved gradients unit, the type of backpropagation and result sharing process as in Fig. 3, is classified into one of three backup methods: layer-wise, neuron-wise, or link-wise backup.

1) Layer-wise (Fig. 3(a)): Only if all of transmissions from layer L to $(L-1)$ are successful, layer $(L-1)$ saves its gradients. When there is any single disconnection among adjacent layers, it loads the previously saved gradients and use them to update the corresponding weights.
2) Neuron-wise (Fig. 3(b)): When all of the transmissions to a single neuron are successful, then the neuron saves its own gradients regardless of the other neurons in the same layer. If neuron A receives the loss from all the neurons in layer L, it computes new gradients.
3) Link-wise (Fig. 3(c)): We save the gradients for each neuron, similarly to the neuron-wise approach. However, with the link-wise approach, if any transmission to the precedent layer is successful, the neuron saves the gradients by substituting the missing value of neuron D to A as zero.

Our layer-wise approach produces a complete value, but it is almost impossible to succeed in the full path in a volatile network. On the other hand, more probable link-wise refreshment gives more chances to save and reuse the former gradients. It should be noted that only after the results are shared at the output layer, we are able to obtain the final loss so that we can start the backward process of the neurons. Therefore, only the precise methods such as the layer-wise and neuron-wise backpropagation except for link-wise backpropagation can be leveraged in the result sharing. According to classification, we aim to use the most sophisticated approach to update as late as possible by using link-wise and neuron-wise result sharing.

However, continuous failures especially in horizontal partitioning and fragile networks, which incur a high possibility of disconnection, force the model to use the same gradients. The updated weights can be calculated as $(W_{i,j} - \gamma \cdot \Delta_{i,j})$, where γ is the learning rate, and training with previous values means that γ increases proportionally with the number of reusage. If we aggressively use the outdated gradient without pause, it entails a high risk of being diverged in learning. Therefore, we restrain the number of reusage to prevent excessively repetitive usage of the saved gradients by introducing $gradReuse_{bw}$ in backpropagation. When the transmission has been unsuccessful during $gradReuse_{bw}$ steps, we do not reuse the saved gradients any longer. If the model gets the updated gradients before $gradReuse_{bw}$, we refresh the number to zero.

However, it is known that this has some risk of divergence [13]. To remedy the divergence problem, we define $gradReuse_{bw}$ to control the learning speed, where $t \leq gradReuse_{bw}$. Since it is hard to directly configure the optimal learning rate, we dynamically adjust $gradReuse_{bw}$, which is similar to the dynamic learning rate tuning [14].

C. Dynamic Update

With either too large $thres_{fw}$ or too small $gradReuse_{bw}$, the weights would be rarely updated, making a model hard to converge, while the opposite cases can allow the weight to be updated even when the success rate is low. Thus, we dynamically adjust these values throughout learning process to make training faster and more stable [9], [15]. During the early stage of training, for a faster saturation of model, the weight should be updated more frequently with smaller $thres_{fw}$ and larger $gradReuse_{bw}$. Then, we monitor the convergence based on an early stopping mechanism, which is to check whether the train loss becomes worse than the currently best model so far. Once the train loss stops improving, we dynamically adjust $thres_{fw}$ and $gradReuse_{bw}$ to be larger and smaller, respectively. In this way, it is possible to construct a rigorous model with credible propagation rather than fast convergence.

V. Runtime Rearrangement against Long-Term Volatility

Due to the intrinsic fluctuation in a wireless edge network, run-time network conditions change over time, possibly resulting in unexpected learning progress far from the intended model. When the whole or partial device connectivity changes due to network configuration variation or obstacle obstruction, it highly affects the overall frequency of disconnections in the learning passes. It means that some more active management approaches for long-term network changes need to be incorporated behind the instantaneous output approximation.

We propose runtime rearrangement to manage the long-term failures in addition to the base structure. Our framework consistently monitors the real-time network fluctuation and adaptively reorganizes the allocated partial models on the workers. Based on the runtime wireless transmission records, we penalize a certain device with poor connections and handover its responsible sub-model to another device from time to time. Then, since the appointed work schedule was determined by the untimely static network connectivity, we also modify the corresponding schedule of the affected devices.

A. Learning Progress Diagnosis for Participating Devices

We propose a cost-effective approach to diagnose the learning progress for long-term connection failure. The training batches are injected in order, and it means that we can perceive the presence of failure without any additional connection check. When a later output has arrived even before an earlier output, it implies that the corresponding earlier batch is lost

TABLE I
THE CREDIBILITY MEASURE AMONG FOUR DEVICES

(a) Initial network connectivity at $t = 0$

Dev. ID	*1*	*2*	*3*	*4*
1	1.0	0.9	0.3	0.5
2	0.6	1.0	0.9	1.0
3	0.3	0.8	1.0	0.8
4	0.5	1.0	0.9	1.0

(b) Runtime network connectivity at $t = 1$

Dev. ID	*1*	*2*	*3*	*4*
1	1.0	0.0	0.5	0.1
2	0.0	1.0	0.0	0.0
3	0.6	0.0	1.0	0.3
4	0.0	0.0	0.2	0.1

(c) Credibility at $t = 1$ when α is 0.9

Dev. ID	*1*	*2*	*3*	*4*
1	1.00	0.09	0.48	0.14
2	0.06	1.00	0.09	0.10
3	0.57	0.08	1.00	0.35
4	0.05	0.10	0.27	1.00

in the middle of learning passes. By recording the actual transmission success history from the senders at the receiver, it is possible to diagnose both network configuration variation and device malfunctioning at once. If a series of packets never arrive, we can decide the situation as the device malfunction. On the other hand, if some deviations from the expected metrics are observed, it is regarded as a network status change. Through this diagnosis mechanism, we do not settle for the present learning structure and consistently suspect and diagnose the volatility of wireless networks and edge devices for discovering an evolving learning structure.

To capture how well a nearby device transmits the learning progress to its receiver, we introduce a credibility measure from the prearranged senders to their own receiver. The receiver, which takes charge of the following passes (i.e., its subsequent layer in the forward passes and its precedent layer in the backward passes), assesses the senders by recording the delivery result. In this phase, some new devices can be discovered, and the credibility of the joining resource is recorded with zero. The newly joined devices are inserted into a layer-wise set with the least number of devices. Since the instantaneous decision may spoil the whole process, we use the exponentially weighted moving average (EWMA) by reflecting the training progress to prevent the surveillance from being excessively controlled by a temporary network status change. We define a credibility measured denoted by $C_s^r(t)$, at time t by collecting the success ratio, for every management window w, for each pair of sender s and receiver r as follows:

$$C_s^r(t) = \alpha \cdot R_s^r[t-w+1, t] + (1-\alpha) \cdot C_s^r(t-w) \tag{4}$$

where $C_s^r(0)$ is initial network connectivity, and $R_s^r[t-w+1, t]$ denotes for the rate of successful delivery during the past w window period from the current time step t. α reflects the weight on the recent connectivity based on EWMA: a smaller α value prioritizes a historical measure relatively more than instantaneous delivery results. With a larger α value, the system can adapt to the network dynamics more aggressively in an early stage. As a side effect, if the network condition frequently changes, it can raise a high migration overhead. In case of the α value of 1, the credibility only reflects the recent communication records.

For example, with four devices and the initial connectivity of Table I(a) and the next runtime connectivity of Table I(b), the credibility is calculated as in Table I(c) using the α value of 0.9. If a certain device malfunctions such as *Dev. 2* in Table I(b), its connectivity with other devices becomes zero. Then, the credibility of the malfunctioned node becomes low, and thus *Dev. 2* is diagnosed as failed.

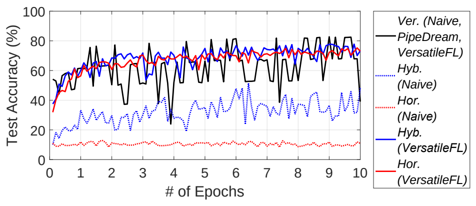
B. Model Rearrangement

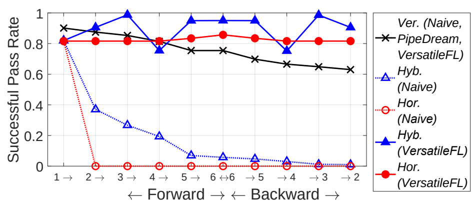
Once we detect some severe dynamics in the network, i.e., the credibility measure of a specific device is below a certain threshold, we adaptively rearrange the current model partitioning for the affected device, to assure durable learning. Since the connectivity of the devices across layers becomes very weak during a certain amount of time, we consider the device as unavailable to perform a given assigned learning task. To tackle the problem, we find a substitutional device to which some neurons from the poor device are reassigned on behalf of it, to keep the learning progress intact. The round-trip in the learning process cannot be reversed in order from the input layer to the output layer in forward and the opposite order in backward. It means that if the rearranged sub-models cross over layers, the full work schedule should also be revised. To make the runtime adjustment defensive yet stable, we assign the neurons to be passed only to the devices in charge of the same layer. It ensures that the rearrangement is constrained within the affected layer only internally.

Given the credibility for the pairs of sender and receiver in each layer, we combine all assessments for a device as the sender and the receiver on average in that the status as both the sender and the receiver affects the net connectivity. For example in Table I(c), the connectivity of *Dev. 2* as a sender is 0.06, 0.09, and 0.10 (to *Dev. 1, 3*, and *4*, respectively), and the connectivity as a receiver is 0.09, 0.08, and 0.10 (from *Dev. 1, 3*, and *4*). Thus, the average credibility of *Dev. 2*, C_2, is given by $avg(0.06, 0.09, 0.10, 0.09, 0.08, 0.10) \approx 0.087$.

When a certain sender device satisfies the condition, $C_s(t) < C_{thres}$, we start rearranging the layer-wise neuron allocation according to the ratio of credibility value. For example, in case of two devices in charge of five neurons each in the layer of $N_{neuron} = 10$ in total, one device has the credibility of [0.1, 0.2], which is measured at each receiver side, resulting in $C_1 = 0.15$ on average, whereas another device has the credibility of $C_2 = 0.6$. The credibility ratio of two devices is calculated as $0.15 : 0.6 = 1 : 4$. Based on the credibility, we set a desirable number of allocated neurons as $neuronNum = [2, 8]$ by $(\frac{1}{1+4} \times 10) : (\frac{4}{1+4} \times 10) = 2 : 8$. By computing the difference of the desirable model partitioning, $neuronNum$, and the current model, $prevNeuronNum$, we can find out how many neurons need to be transferred from one to another, $neuronMove = [-3, 3]$. Then, *Dev. 1* randomly selects three neurons and passes them to *Dev. 2*.

We reorganize model partitioning across all layers and all workers. If the whole network changes, all of the initial processes have to be done from the beginning. Then, the work schedule of each worker should also be newly allocated at


(a) Test accuracy with respect to the training epochs


(b) Successful pass rate in layer-to-layer propagations

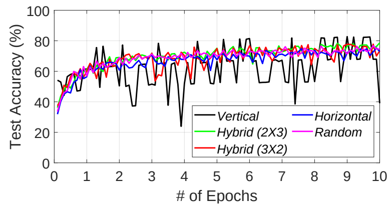

(c) Different model partitioning

Fig. 4. Performance of *VersatileFL* with the short-term link fluctuation according to each different model partitioning structure

runtime, since the given schedule has been determined by the initial network connectivity. However, generating a totally new schedule across the whole model may incur additional costs due to its resulting computation overhead to manage the new schedule. Therefore, in our adjustment scheme, only the devices with connectivity changes participate in the rearrangement process. We consider backup training for both expected and unexpected failures. When the devices are pre-conscious of their incoming breakdown for predictable situations such as battery outages, the trained parameters in the allocated sub-model are handed over to a selected target device. However, if a participating device is missing in some unpredictable accidents, the newly assigned partition taken over from the missing device is trained from the beginning.

Through local management, we leave the devices with stable connectivity unaffected by the network condition and process their own primary tasks. Moreover, due to the fixed sequence of round-trip, where the forward and backward passes are enforced wholly in the reverse order, we convey the records in the outputs. Since all devices in charge of adjacent layers can inter-communicate in DNNs, we readily inform the progress of adjustment. In this way, the schedule is maintained in the order for the overall schedule while proceeding with all of the batches with the rearrangement on each layer itself.

Since more devices participate in each layer towards *Horizontal* allocation, the communication overhead increases. It means that even though we load the neurons into other devices, the number of communications for learning is constant. Moreover, when there are affordable devices, we can prevent the excessive transmission cost by inserting the incoming devices into the layer-wise set of the least number of devices.

VI. Evaluation

We validated *VersatileFL* using *PyTorch* 1.4.0 with *Python* 3.7. We conducted base experiments with DNNs of six dense layers including four hidden layers with 128 neurons. We used the activation function of Rectified Linear Unit (ReLU) and the basic gradient descent optimizer, which do not need any additional intra-layer information. We set the learning rate to 0.01 and the batch size to 100. We used *MNIST*, *EMNIST*, *CIFAR-10*, and mainly *Fashion-MNIST* dataset consisting of 60,000 train data and 10,000 validation data. Considering the characteristics of the dataset, the loss function is defined to be softmax cross-entropy with logits. To prevent the performance ruled by the initial weights, we investigate resilience performance under five different trials.

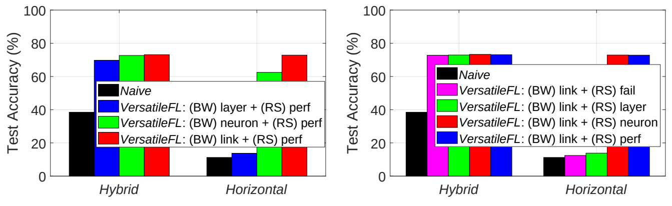
We simulated a wireless network of six edge devices over the Region of Interest (RoI) of $50 \times 50m^2$ using the combined path-loss shadowing model. We varied the path-loss exponent from 3.0 to 3.6, and 3.2 is mostly used as the default setup. To simulate a long-term critical link failure scenario, the path-loss exponent of 6.0 is used. The reference loss is set to 46.68dB, and the white Gaussian noise of $\mathcal{N}(0, 8^2)$. We adopted the Packet Reception ratio (PRR) as the main connectivity measure. For the initial network, we counted the percentage of successful transmissions by sending 50 packets and quantified the overall link connectivity of 77.67 % on average. To obtain the statistically meaningful results on top of the dynamic fluctuation, we ran five training experiments with 100 test runs per training and took the average performance.

We partitioned DNNs into six sub-model shards such that six edge devices are assigned to their own sub-model. We implemented four different partitioned models: 1) *Vertical*: a single device is responsible for a whole layer based on the core partitioning architecture and 1-Forward-1-Backward scheduling from *PipeDream* [16]; 2) *Horizontal*: a single device participates across all of the layers, which is an extreme opposite with *Vertical*; 3) *Hybrid*: a compromise between *Vertical* and *Horizontal* to partition in the level of both layers and neurons that borrows the resilient partitioning mechanism under volatility from *EdgePipe* [10]; and 4) *Random*: a random distribution over the devices, but the allocated neurons per worker show almost similar to *Horizontal* due to the regularity of the randomness. For load balancing, we evenly partition except for *Random*. Among various work scheduling schemes, we borrow the idea of an efficient device work schedule from *EdgePipe* [10], which accelerates the learning process by applying the pipeline schedule into model parallelism. We have compared our scheme with *Naive*, which does not have any maintenance scheme against failures.

In *VersatileFL*, we tuned the parameters of $thres_{fw}^{MAX}$ to be 0.5, which means we consider the forward as valid when the number of connected links is larger than the number of missing signals. $thres_{fw}$ varies from 0 up to 0.5 by increasing in the interval of 0.1 as the loss decreases for 60 steps. For backpropagation, we used the $gradReuse_{bw}^{MAX}$ value of 10 with the decrement level of 1 down to 0. In addition, there is a possibility that some more generalized tuning approaches considering the network connectivity can be embedded.

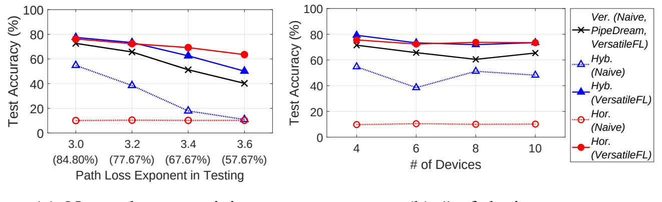
A. Resilience under Short-Term Link Fluctuation

First, we investigated resilience learning performance using various partitioning models, without long term-wise manage-



(a) Backpropagation (b) Result sharing

Fig. 5. *VersatileFL* with respect to the granularity of backpropagation with the ideal result sharing and result sharing with the link-wise backpropagation in horizontal allocation



(a) Network connectivity (b) # of devices

Fig. 6. *VersatileFL* in various network configurations

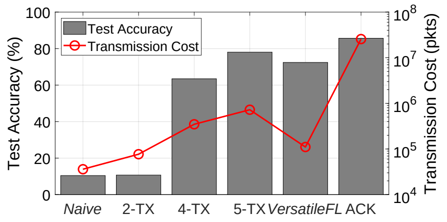


Fig. 7. Test accuracy and transmission cost depending on different transmission failure recovery schemes

ment in Fig. 4. As shown in Fig. 4(a), the *Horizontal* allocation in which more distinct devices are allocated to each layer, benefits more from volatile learning. Without sub-optimally training the model, there is no perfect data delivery until the output layer in horizontal allocation at all, which means it failed to converge due to the high risk of disconnection between adjacent layers. On the other hand, when we partition DNN vertically in the level of layers, only single transmission is needed for propagation to the subsequent layer, resulting in fast convergence with a maximum test accuracy of 83.11 %. However, since the *Vertical* allocation is more likely to lose the whole output due to a single failure, it shows just a binary inference result: totally successful or totally failed delivery.

Taking a closer look at the rate of successful passes in Fig. 4(b), the volatile learning approach takes advantage of less probability of losing the whole layer in horizontal allocation and shows the most stable performance with the lower spikes. The cascading propagation degrades the hostile learning with a dramatic decrease, whereas horizontal allocation in our scheme rescues the successful delivery in the last layer-to-layer backpropagation. It implies that *VersatileFL* contributes to making the learning path resilient under the link uncertainty.

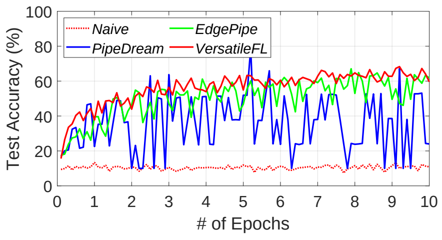
In Fig. 4(c), different partition models including *Hybrid* (3×2) with two devices per layer across three consecutive layers are compared to each other. We verified that *VersatileFL* performs well in various model partitioning. Even though we randomly partition the neurons considering neither load balancing nor volatility, *Random* partitioning can achieve the almost same performance as others. It means that our management makes any partition types for distributed learning *versatile* to the turbulent network dynamics.

We examined how the granularity of learning affects the overall performance in the aspects of backward passes and result sharing in the horizontal model partitioning in Fig. 5. As indicated in Fig. 5(a), with the assumption that there is no failure upon calculating the loss in the result sharing phase, the most strict layer-wise updates are almost untrained, since any single transmission failure can spoil the learning path. When we scale down to neuron-wise and link-wise approximation, we can successfully backpropagate with the partial paths. The horizontal allocation fully benefits from versatile learning and achieves 51.24 % and 61.61 % improvement in neuron-wise and link-wise, respectively. On the other hand, the hybrid allocation allocates two devices to one layer, and all of the results show almost the same performance improvement.

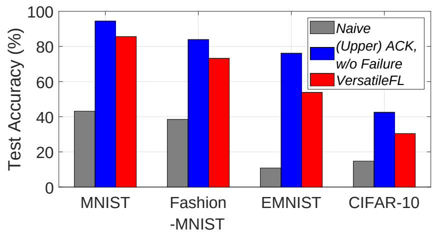
Regarding the result sharing in Fig. 5(b), we investigated the effect of volatility at the last result sharing step in the DNN architecture. We compared the realistic intermittent transmission with the ideal perfect transmission at the result sharing step. We applied link-wise backpropagation in the experiments. We obtain the similar trend with the backpropagation granularity. A narrow approach contributed to early convergence with a small gap against the ideal result sharing case. With our neuron-wise updates in result sharing, we can achieve successful training as the ideal result sharing case under volatility. From Fig. 5, if we do not consider the sub-optimal updates in learning, the whole training can be collapsed even with a single transmission failure.

In Fig. 6, we investigated the effect of network connectivity by varying the path-loss exponent and the number of devices participating in learning. As the network deteriorates as shown in Fig. 6(a), the learning path has a high risk of collapse, and *Naive* without managing the link volatility seriously suffers from learning. Although our versatile learning also degrades, we maintain relatively stable training progress with only a 12.71 % decrease in *Horizontal* even when the network connectivity of averaged PRR varies from 84.80 % to 57.67 %. Even under severely broken networks, *VersatileFL* takes advantage of the sub-optimal training, resulting in a performance gap of 66.16 % and 22.65 % compared to *Naive* in horizontal and hybrid allocation, respectively. We varied the number of devices over the fixed six layers in Fig. 6(b). *VersatileFL* shows stable learning with more workers, incurring a higher probability of disconnection from the more transmissions.

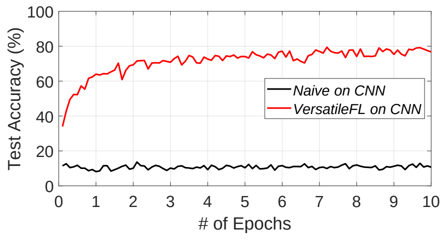
We compared our *VersatileFL* with respect to inference accuracy and transmission cost with two recovery schemes without versatile management: 1) *r-TX*: sending the same packet with multiple *r*-times consecutively; and 2) *ACK*: keeping sending the packet until receiving its acknowledgment. As indicated in Fig. 7, we reached 61.96 % higher testing accuracy than *Naive* with a factor of 6 in communication overhead, which is similar to a consecutive packet transmission with four times. Interestingly, *VersatileFL* shows the best trade-



(a) Different federated learning approaches in the wireless networks with a path loss exponent of 3.6

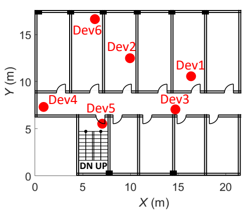


(b) Different datasets

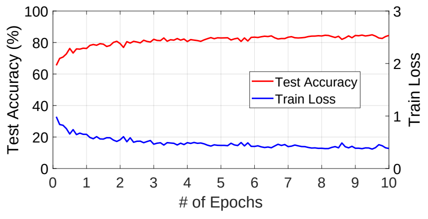


(c) Different model of a convolutional network

Fig. 8. Test accuracy of various learning settings

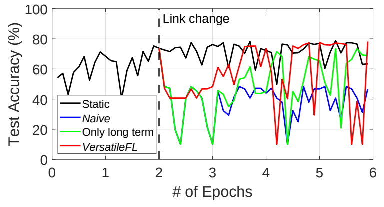


(a) Device distribution

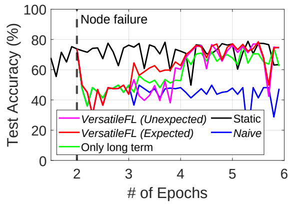


(b) Test accuracy

Fig. 9. Validation in a real-world wireless testbed



(a) Long-term link connectivity variation of *Dev. 2*



(b) Long-term failure, where the *Dev. 4* is malfunctioning

Fig. 10. Runtime rearrangement dynamics when the network configurations are varied at epoch 2

off performance considering both accuracy and cost compared to *r-TX*. Our scheme enables training even with the partially delivered data. As another interesting result, *VersatileFL* shows competitive inference accuracy, compared to *ACK*, achieving a similar performance with the wired connection using the excessive transmission cost, compared to ours.

We validated *VersatileFL* in a relatively poorer network environment in Fig. 8(a), compared to 1) *Naive* without recovery mechanisms; 2) *PipeDream* [16]: a federated learning with vertical model parallelism in wireless network scenarios; and *EdgePipe* [10]: a distributed learning designed for wireless edge devices. We verified that our maintenance scheme plays a key versatile role in volatile learning environments, outperforming other distributed learning in wireless networks. Interestingly, *VersatileFL* also outperforms *EdgePipe* in the aspect of the learning speed, by means of our sub-optimal training with the saved gradients against the broken propagation.

We verified the feasibility of *VersatileFL* under volatility to train various datasets with MNIST, Fashion-MNIST, EMNIST, and CIFAR-10. As illustrated in Fig. 8(b), since the horizontal model under the network fluctuation is collapsed even with single disconnection, *VersatileFL* becomes quickly fragile both in learning and inference across all of dataset. However, our maintenance scheme makes learning viable up to 85.63 %, 73.30 %, 53.89 %, and 30.45 %. Taking a closer look, under perfect data delivery without network failures, ours achieves almost similar performance with a factor of 0.91, 0.87, 0.71, and 0.75, respectively. It means that *VersatileFL* keeps stable learning against unexpected network variations.

From Figs. 7 and 8(b), it is obvious that the learning speed or quality of the federating learning in an ideal environment without any propagation failures outperforms those of ours. However, as pointed out in Fig. 7, in the volatile wireless medium, excessive transmissions are required to meet the similar performance in case of the wired stable connectivity. As a trade-off between the learning performance and the communication overhead, *VersatileFL* offers a reliable yet efficient way considering both performance and resource budget.

We validated the efficacy of the maintenance scheme based on a different neural network model of convolutional neural network (CNN). We constructed a simple CNN architecture consisting of one convolutional layer and one pooling layer in front of the four dense layers. As illustrated in Fig. 8(c), *Naive* failed to proceed the learning, whereas *VersatileFL* achieved relatively high stable performance, reaching up to 80 %. It implies that our maintenance scheme against the learning pass failures has indeed helped various neural network models to make efficient and stable progress via runtime versatile learning and model rearrangement.

We verified *VersatileFL* in a real-world testbed. We used six TelosB motes with IEEE 802.15.4 wireless radio over the RoI of $20 \times 20m^2$ in Fig. 9(a). The average PRR of 1,000 packets in the network is measured to be 71.62 %. As shown in Fig. 9(b), *VersatileFL* achieves stable learning with more than 80 % accuracy with Fashion-MNIST. It means that *VersatileFL* is validated in real-world wireless radio environments.

B. Maintenance against Long-Term Network Dynamics

For rearrangement, the management window w, is one epoch (i.e., 600 steps). Regarding the credibility-related parameters, α is set to 0.9, and the C_{thres} is set to 0.7767, which is the overall averaged PRR in the initial network setup. It means that we start adjusting the model when the runtime connectivity is different from the given network status.

We examined how *VersatileFL* monitors and survives against long-term variations in Fig. 10. At epoch 2, we varied the network condition of partial links, where the link connectivity from all devices to *Dev. 2* and from *Dev. 2* to all devices are deteriorated from the exponent of 3.2 to 6.0, in Fig. 10(a) and device malfunctioning in Fig. 10(b). In both results, our surveillance system records the history, and it observed that some network variations occurred during epoch 2. It is

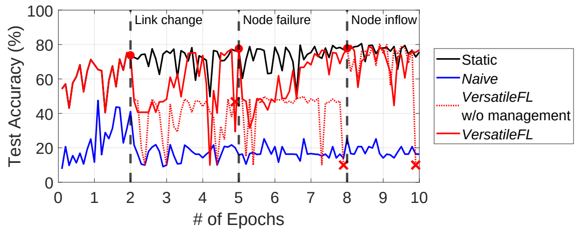


Fig. 11. Stable learning against short-term and long-term network fluctuations, where network variations occur at the red circled points

interesting to observe that at epoch 3, the credibility measure internally detects the severely degraded network condition, and *VersatileFL* steadily recovers the learning path through the model arrangement. Another interesting point is that if the runtime model arrangement is augmented with the short-term link failure backup scheme, it leads to even faster restoration.

We also investigated the recovery performance depending on the type of device failures in Fig. 10(b): 1) *Unexpected*: a device can malfunction unexpectedly, and thus the trained submodel is all lost; 2) *Expected*: a device can be aware of its vitality, and the trained parameters are passed to other devices before failure. Intuitively, the *unexpected* failure requires a submodel retraining from the beginning, spending more time to recover a valid learning model. Although the learned parameters are missing, our backup scheme provides a way to restore a valid model both in *expected* and *unexpected* cases.

We investigated how *VersatileFL* adapts to the dynamic network change, in Fig. 11. The intermittent link fluctuates all the time throughout the training, and the partial links and a single node deteriorate at epoch 2 and epoch 5, respectively. We also compared to *VersatileFL* without management, which is same as applying *Naive* to the *VersatileFL-based* trained or recovered model after each network variation outbreak. Without any failure-against management, the existing deep learning frameworks cannot survive under hostile environments from the beginning. It implies that although we already construct the model using *VersatileFL*, further perpetual management is essential. At worst, the training process is totally blocked only with single device malfunctioning. However, *VersatileFL* keeps almost the same training performance compared to *Static* even with not only intermittent links volatility but also long-term changes. Moreover, when a single node joins at epoch 8, *VersatileFL* embraces the newly added resource and hands over a partial model to the device. With the relaxed resources, we keep stable learning and inference, while each device takes charge of the less workload.

VII. Conclusion

We have presented *VersatileFL*, a novel deep learning framework under volatile wireless networks. To solve the short-term and long-term volatility problem, we suggest two maintenance schemes: 1) versatile distributed learning for the short-term fluctuation; and 2) runtime rearrangement for the long-term network failure. First, to enable learning under intermittent link disconnection, we approximate the missing intermediate values in both forward and backward passes by ignoring the unknown signals or substituting the previous measure. We further propose a model adjustment by diagnosing the runtime connectivity and rearranging the sub-model shards in proportion to the credibility of each worker. We have demonstrated that our volatile distributed learning makes distributed training resilient under link fluctuations. Moreover, our recovery scheme of model rearrangement captures the variation in network configurations and adaptively improves the model along with the runtime condition.

For future work, it would be interesting to combine data parallelism with our model parallelism-driven edge learning in a hybrid manner. Depending on the degree of priority and importance between computation resource and data generation, we may discover a more dynamic practical way to perform distributed learning in the edge network setting.

References

[1] L. Liu, J. Zhang, S. Song, and K. B. Letaief, "Client-edge-cloud hierarchical federated learning," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–6.

[2] H. Sun, S. Li, F. R. Yu, Q. Qi, J. Wang, and J. Liao, "Toward communication-efficient federated learning in the internet of things with edge computing," *IEEE Internet of Things Journal*, vol. 7, no. 11, pp. 11 053–11 067, 2020.

[3] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *arXiv preprint arXiv:1909.08053*, 2019.

[4] J. Lee, J. Cho, and H. Lee, "Stitchnet: Distributed on-device model partitioning over edge devices under volatile wireless links," *IEEE Access*, vol. 10, pp. 110 616–110 627, 2022.

[5] M. Chen, D. Gündüz, K. Huang, W. Saad, M. Bennis, A. V. Feljan, and H. V. Poor, "Distributed learning in wireless networks: Recent progress and future challenges," *IEEE Journal on Selected Areas in Communications*, 2021.

[6] M. Chen, H. V. Poor, W. Saad, and S. Cui, "Wireless communications for collaborative federated learning," *IEEE Communications Magazine*, vol. 58, no. 12, pp. 48–54, 2020.

[7] S. Pal, E. Ebrahimi, A. Zulfiqar, Y. Fu, V. Zhang, S. Migacz, D. Nellans, and P. Gupta, "Optimizing multi-gpu parallelization strategies for deep learning training," *IEEE Micro*, vol. 39, no. 5, pp. 91–101, 2019.

[8] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[9] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, 2020.

[10] J. Yoon, Y. Byeon, J. Kim, and H. Lee, "Edgepipe: Tailoring pipeline parallelism with deep neural networks for volatile wireless edge devices," *IEEE Internet of Things Journal*, 2021.

[11] J. Chen, X. Pan, R. Monga, S. Bengio, and R. Jozefowicz, "Revisiting distributed synchronous sgd," *arXiv preprint arXiv:1604.00981*, 2016.

[12] D. Gong, Z. Zhang, Q. Shi, A. van den Hengel, C. Shen, and Y. Zhang, "Learning deep gradient descent optimization for image deconvolution," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 12, pp. 5468–5482, 2020.

[13] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[14] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[15] H. H. Yang, Z. Liu, T. Q. Quek, and H. V. Poor, "Scheduling policies for federated learning in wireless networks," *IEEE transactions on communications*, vol. 68, no. 1, pp. 317–333, 2019.

[16] D. Narayanan, A. Harlap, A. Phanishayee, V. Seshadri, N. R. Devanur, G. R. Ganger, P. B. Gibbons, and M. Zaharia, "Pipedream: generalized pipeline parallelism for dnn training," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, 2019, pp. 1–15.