

## RESEARCH ARTICLE

# StitchNet: Distributed On-Device Model Partitioning Over Edge Devices Under Volatile Wireless Links

JIHO LEE<sup>1</sup>, JEIHEE CHO<sup>2</sup>, AND HYUNGJUNE LEE<sup>1</sup> , (Member, IEEE)<sup>1</sup>Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA<sup>2</sup>Department of Computer Science and Engineering, Ewha Womans University, Seoul 03760, South Korea

Corresponding author: Hyungjune Lee (hyungjune.lee@ewha.ac.kr)


This work was supported in part by the National Research Foundation of Korea (NRF) funded by the Korean Government (MSIT) under Grant NRF-2021R1A2B5B01002906; and in part by the Institute of Information, Communications Technology Planning and Evaluation (IITP) funded by the Korean Government (MSIT) for the Artificial Intelligence Convergence Innovation Human Resources Development (Ewha Womans University) under Grant RS-2022-00155966.

**ABSTRACT** Distributed deep learning architecture can achieve scalable learning and inference capability at resource-constrained edge devices. Although the parallelization-based approaches have actively been investigated in the edge computing context, they are not designed for the devices that are usually wireless and mobile, causing the substantial link and device failure issue. We propose a semi-distributed deep learning architecture, *StitchNet*, based on model parallelism for volatile wireless edge networks. Our algorithm first classifies a set of effective neurons with a substantial impact on their connected neurons across layers. Then, an opportunistic neuron assignment is employed to ensure the full forward and backward propagation paths by stitching the subsets of the model across the devices with path redundancy via *neuron cloning* for securing high resilience to network and device uncertainty. Simulation-based experiments demonstrate that *StitchNet* has achieved high inference quality on visual classification tasks even under the volatile lossy network environment, by making edge devices collaboratively find and stitch their in-ward and out-ward edge paths with a reasonable communication overhead.

**INDEX TERMS** On-device AI, distributed learning, model parallelism, edge computing.

## I. INTRODUCTION

Due to the great advances in deep learning over the last decade particularly in several domains such as image classification [1], [2], natural language processing [3], machine translation [4] to reinforcement learning [5], the deep learning-based approach has been regarded as a popular system design methodology. Due to its outstanding generalization property acquired by the increasing model size and the huge amount of training data, complicated algorithms or computer systems can be built relatively easily and quickly without extracting the underlying features in a hand-crafted manner.

The associate editor coordinating the review of this manuscript and approving it for publication was Hosam El-Ocla .

As the edge computing technology allows to bring computation and data storage to the edge side, such as mobile phones, sensors or wearable devices, the problem of scalable deep learning on distributed infrastructures has recently been investigated [6], [7]. Traditional centralized deep learning suffers from large communication overhead and privacy breaches caused by uploading raw data from the user side to a central computation server. Distributed system architecture can offer a practically feasible way to achieve scalable learning and inference capability at resource-constrained edge devices.

In order to make distributed deep learning available at the edge, the concept of collaborative learning, also called *federated learning*, has been proposed [8], [9], [10]. Parallelizing learning and inference tasks into multiple devices is classified into two categories: data parallelism and model parallelism.

In data parallelism, each of devices trains the same replicated learning model, while using a different mini-batch training data from itself, and aggregates the gradients from all replicas to update the model parameters [2]. Model parallelism, on the other hand, splits the model into its sub-parts across separate devices for scalability, while using the same training data batch at devices [4], [11], [12], [13], [14], [15], [16].

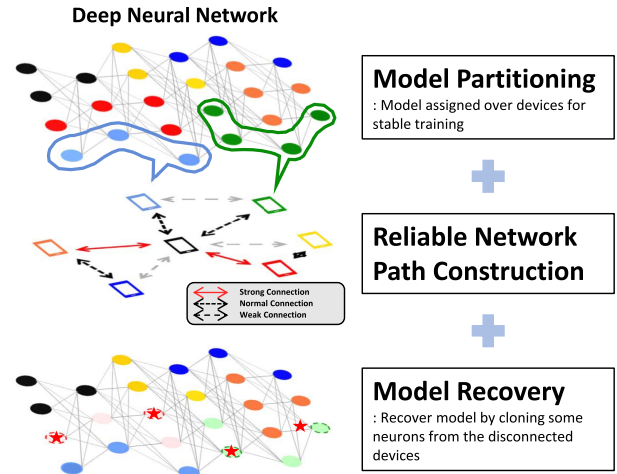
Particularly in the on-device context, several distributed learning approaches such as Federated Split Learning [17] and FedMask [18] have been proposed. FedMask [18] is a federated learning-based approach [8] with mobile devices by considering both communication and computation efficiency based on efficient sparse binary masking. Federated Split Learning (FSL) [17] is a hybrid learning architecture that combines Federated Learning [8] and Split Learning [19] rather with the focus of privacy awareness.

Although the distributed deep learning approaches via model parallelism suggest a desirable paradigm for scalable learning and inference under the memory constrained edge environments, they explicitly or implicitly assume a lossless network in which cross-device communication is reliable without any lossy link failures, or by relying on a dedicated transport layer. In general, however, most of edge devices become mobile and more resource-constrained with a lightweight transport layer. Designing a distributed learning architecture for volatile wireless networked devices is an even more challenging problem.

In this paper, we first propose a semi-distributed deep learning architecture, *StitchNet*, based on model parallelism for volatile wireless edge networks. In a lossy network formed by wireless edge devices, a device coordinator, which is selected among edge devices, performs model partitioning. A different subset of neurons at layers is assigned to the devices, depending on device-to-device link uncertainty and workload at the device. This assignment is made by taking into account the requirement that the forward and backward passes should be established by *stitching* the subsets of the model across the network, while minimizing the intermediate edge loss in the middle of the learning paths.

Once a deep neural network is partitioned into several sub-models, a sub-model is assigned to its dedicated edge device considering link quality between devices. Further, in order to fight against critical or permanent device failures, *StitchNet* performs a *local* rearrangement via *neuron cloning* with consideration of the impact of neurons, borrowing a concept from the pruning and sparsification [20]. This ensures to attain a complete learning path across layers even under the unreliable networks.

Our approach proposes a new on-device model partitioning technique. It exploits opportunistic computation through some of redundant neuron-to-device assignments to compensate for the learning path losses caused by the lossy device-to-device link failures. *StitchNet* is capable of capturing all possible learning update information on some important neurons from multiple devices, along both the forward and backward paths.



**FIGURE 1.** High-level description of our proposed distributed on-device learning architecture, *StitchNet*, where the colored circle represents the neuron that are allocated to and computed by their corresponding device with the same color, where gray colored connection is not used while learning.

The main contributions of this work can be summarized as follows:

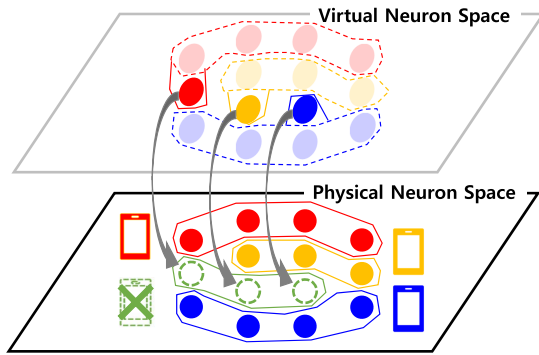
- We present a semi-distributed deep learning architecture based on model parallelism, which is resilient against a volatile lossy network consisting of wireless edge devices.
- We offer a novel neural network partitioning scheme called *Mix-Mapped* partitioning that allows to form an efficient learning architecture with decent communication cost and stable accuracy performance.
- We suggest an opportunistic neuron assignment method that establishes the full forward and backward propagation paths by stitching a subset of the model across the network with path redundancy via *neuron cloning*, fighting against the uncertainty over the otherwise fragile learning path.

This paper is organized as follows: After describing the system model in Sec. II, we present our model partitioning scheme in Sec. III and our model recovery scheme in Sec. IV. We validate this work with various real-world datasets in Sec. V and then conclude our work in Sec. VI.

## II. SYSTEM MODEL

We consider a new way of implementing a distributed deep learning model on edge devices based on model parallelism, under the wireless lossy device-to-device link environments, as illustrated in Fig. 1. In volatile wireless networks, the recent state-of-the-art distributed deep learning approaches based on model parallelism may not work any longer. Under the link or/and device failure situations, the intermediate computation result performed by one device may be likely lost at the other device in charge of its following computation in the middle of the learning process.

To tackle the volatility of distributed neural network architecture caused by that of link connectivity, we apply the



**FIGURE 2.** Virtualization of neurons across layers, placed at a physical neuron space and a virtual neuron space in *StitchNet*. In case of device malfunction (e.g., the green device), the neurons under the primary device are reallocated to one or more secondary devices.

concept of virtualization to neurons across layers, which are supposed to be computed by physical devices. The neurons can be defined the ones in two spaces: physical neuron space and virtual neuron space, as shown in Fig. 2. A neuron belongs to both spaces, and can be allocated to a physical device through each space.

The physical device at one space may well be different from that at the other space. In reality, such as in the lossy network scenario, in case of the temporary (e.g., due to link loss) or permanent (e.g., due to device loss) failure on the primary device, a different physical device as the secondary role needs to seamlessly take over the interrupted task. Incorporating this functionality can greatly help a distributed learning architecture to sustain the learning and inference capability.

In this work, we assume that edge devices are equipped with a wireless radio interface (e.g., 802.11 WiFi, or LTE/5G cellular), and are deployed in a stationary environment where they can form a connected edge network. We consider a scenario that multiple edge devices are connected and cooperate to decide the status of the environment or persons such as patients in hospital. The information can be gathered from edge devices with sensors, while training a model using newly collected data. For this work, it is assumed that the devices involved in learning process are credible so that there is no privacy issue on sharing information to other devices.

A communication-centric device that has the highest connectivity to the other devices is selected as a device coordinator. The device coordinator manages the model partitioning initially once or in case of critical loss or device failures. From the perspective of task computation, the device coordinator initiates the first input layer, aggregates and distributes the intermediate results from one device to the other, and processes the last layer. In this work, it is assumed that all of the edge devices including the device coordinator are homogeneous.

To solve the problem of model partitioning in a lossy wireless network, we introduce a dynamic neuron-to-device mapping mechanism called *StitchNet*, based on the concept of physical and virtual neuron spaces, with three main phases:

- 1) Mix-Mapped partitioning; 2) path construction; and 3) model recovery via neuron cloning as illustrated in Fig. 1.

### III. MODEL PARTITIONING

We incorporate a model partitioning on a specific neural network architecture by taking into account the characteristics of distributed learning. In wireless network environments, since the results may be lost due to intermittent connection, constructing a stable learning path is crucial to quickly boot up the learning procedure. However, there may be a concern that pursuing accuracy alone can increase communication cost in the network aspect while training. Therefore, we seek a simple yet effective way of partitioning a deep learning model over edge devices.

We introduce a new neural network partitioning scheme called *Mix-Mapped* partitioning, in which a part of the model contributes to increasing the learning performance and the other part benefits reducing communication cost based on device connectivity.

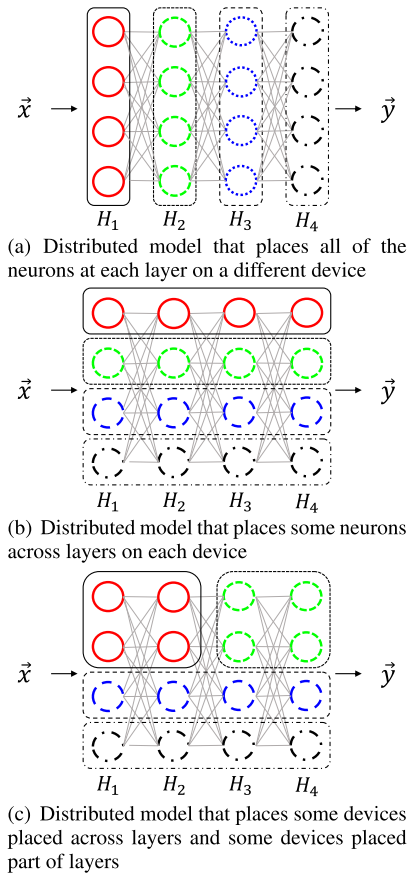
Beyond it, in order to minimize intermittent disconnection during training procedure, a device coordinator with central connectivity is needed, and also direct communication between devices should be able to make the learning path connected, thereby creating an alternative path to ensure a more effective and stable learning path.

We conduct a neuron-based model partitioning in which a single neuron is considered as a base unit for splitting the model. A layer consists of a number of neurons, and a neuron is connected to others neurons located at its preceding and following layers. As necessary notations on neurons and layers,  $h_m^n$  represents the  $m^{\text{th}}$  neuron at the  $n^{\text{th}}$  layer, with its weight values in matrix  $H_n$ . For example, the second hidden layer comprises neurons  $h_1^2$ ,  $h_2^2$ , and  $h_3^2$ , with its weight matrix  $H_2$ ; and  $w_{21}$  in  $H_2$  indicates a weight value of the edge from neuron  $h_2^1$  in the first hidden layer to neuron  $h_1^2$  in the second hidden layer.

#### A. MIX-MAPPED PARTITIONING

During the training epochs in a neural network model, the weight values continue to be adjusted to minimize the loss value at the output layer. Each layer plays a key role in passing out the intermediate computation results to a following layer, and losing a substantial portion of the layer causes a huge risk, particularly in wireless networks. Depending on a different model partitioning as shown in Fig. 3, the lost portion of the layer becomes very different. For a model that places all of the neurons at each layer into one single device, the potential lost portion of the layer would be very high. On the other hand, if the neurons are fairly distributed across layers to a single device, the potential loss risk would be marginal and it may be possible to have the ongoing training almost intact.

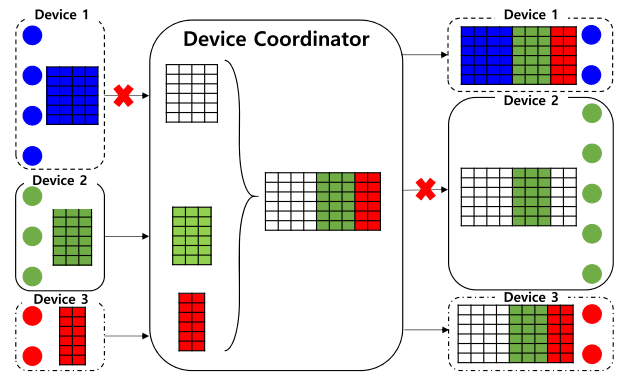
To make a number of devices share the learning results with effective communication, we use the broadcast communication to pass the result into all possible connectable devices. In the vertical partitioning model as shown in Fig. 3(a), only



**FIGURE 3.** Different model partitioning cases where a circle denotes a neuron, marked with a different color on each device.

one communication to one single device in charge of the next layer is sufficient to send the computed result to the next layer. In the horizontal partitioning as shown in Fig. 3(b), on the other hand,  $K \times (K - 1)$  time communications are required where  $K$  denotes the number of devices in charge of one layer. The cost of the horizontal model will increase immensely as the number of devices increases.

Our *Mix-Mapped* partitioning model leverages the advantages of both the vertical model (as in 3(a)) and the horizontal model (as in 3(b)) for the lower communication cost and the stable accuracy performance. To amplify the viable horizontal learning paths, several devices with higher connectivity are allocated for the horizontal learning paths, by finding out an effective set of neuron-to-device mappings on the physical and virtual neuron space, as previously shown in Fig. 2. Some other devices with lower connectivity are allocated for the vertical learning paths so that they can be located in the consecutive layers as nearby as possible. As the low-power wireless interface suitable for edge networks tends to have volatile asymmetric link behaviors [21], the bidirectional packet reception ratio (PRR) is adopted to quantify the device-to-device connectivity. The multiplication of the bi-directional PRR values is calculated as a connectivity measure between two adjacent devices. Once the bidirectional



**FIGURE 4.** Aggregation and distribution of intermediate computation results among devices.

PRRs are calculated on average for all possible link pairs, we allocate a group of devices that has their own PRR value higher than the median of the calculated PRRs for all of the links to the horizontal device group, and the remainder to the vertical device group, as shown in Fig. 3(c).

### B. LEARNING PATH CONSTRUCTION WITH DEVICE COORDINATION

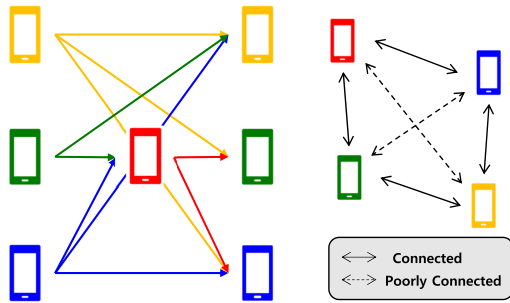
#### 1) DEVICE COORDINATION WITH PARTIAL UPDATES

*StitchNet* performs a device coordination for participating devices. A device can be elected as the device coordinator voluntarily or by election depending on its connectivity to other devices. If a device want to declare itself as the device coordinator, the device coordinator can initiate the model partitioning procedure. Otherwise, the edge devices located within the radio range can elect a cluster head among them, as the device coordinator. Since the problem of electing a cluster head has extensively been investigated in the wireless sensor network community [22], any mature election algorithm can be applied with our model partitioning, depending on a different design requirement, such as energy consumption, node type, and mobility.

Once a model is partitioned, the device coordinator initiates the computation from the input layer, and passes out its result to the devices that are in charge of the neurons belonging to its following layer via *path stitching*. In case of a link failure from the device coordinator to a device, or vice versa, we let it happen. We do not make up for the transmission failure, such as via retransmission.

The device coordinator operates within its own timeslot without synchronizing with other devices. Since it manages the computation aggregation and distribution among devices from one layer to the other, it sends and receives the intermediate results within a fixed timeslot. If some expected results are not received by the end of the expected timeslot, the device coordinator proceeds without any make-up with the problematic device, as shown in Fig. 4.

If a device that does not receive the expected result, it uses the result of the device itself for its subsequent operations on both forward and backward passes. By utilizing the device



**FIGURE 5.** Path construction between devices, selecting the path with more high connectivity. A set of devices take in charge of certain neurons across the layers in a neural network, where connectivity between devices varies.

coordinator and partial result update, each device can update their weights more frequently using the results that could be lost under wireless networks, so that the model can converge quickly.

## 2) PATH CONSTRUCTION

To improve accuracy further, a device not only passes out the results to the device coordinator, but also uses device-to-device communication to use as many learning paths as possible. As shown in Fig. 5, if there is a higher probability that the yellow colored device receives the intermediate results from its nearby devices, we directly use the alternative device-to-device path rather than through the coordinator.

Although the device coordinator plays some critical roles in conducting partitioning and maintaining the model parameters, our framework offers a way for edge devices to keep performing learning process even if the device coordinator is temporarily disconnected under the volatile lossy wireless transmission, operating in a semi-distributed fashion.

## IV. MODEL RECOVERY

Once a neural network model is partitioned into devices based on the mix-mapped partitioning model, we perform a training procedure towards secure and high performance, while preserving communication cost. We want to make the model cope with uncertainty via the *volatile* learning paths, caused by dynamic factors such as device failures in a lossy wireless network.

We evaluate how much the learning can be deteriorated by a device failure, and then perform a neuron-to-device rearrangement procedure via *neuron cloning*. By reflecting the link dynamics in the learning model, the model can compensate for critical link losses at certain devices. Depending on whether the sub-part of the model are close to horizontal or vertical partitioning, we perform cloning the impacted neurons at another neighboring device with the increased computation overhead. It should be noted that our model recovery system occurs when the device has failed for a long time. Model recovery is activated to cope with the permanent

loss of neurons when physical device is not available anymore due to multiple reasons.

### A. LOCAL REARRANGEMENT VIA NEURON CLONING

Even though a steady-state stable performance is reached via priority-based partitioning, some partial link or/and device breakdowns, which frequently occur in a real-world network, can extensively degrade the learning and inference quality.

We address this problem with a local rearrangement via *neuron cloning*. If the device coordinator does not receive any update from a specific device continuously for a certain fixed amount of time, the device coordinator attempts to *locally* rearrange and *clone* the neurons that were previously assigned to the device under failure to one or more other devices randomly selected among the available devices, as shown in Fig. 2.

According to our *Mix-Mapped* scheme, the horizontally partitioned devices play a crucial role to accommodate accuracy by stitching the learning path. If one of those devices fails, the accuracy of the learning model would severely be affected. Therefore, to prevent the loss of accuracy, we clone the entire neurons from the failed devices, and then assign them to available devices.

Once a new device is selected, the recently recorded weight values on the neurons are loaded from the device coordinator. Although the weight values stored at the device coordinator may have been outdated, they still contain some historically crucial information on the edge weight, and can help to make a quick and stable convergence.

### B. NEURON CLONING WITH PRIORITY

Although cloning the entire neurons from the failed devices to backup devices would indeed be helpful to keep the accuracy, it is important to evaluate the effect of a single neuron or a group of neurons on learning efficacy. In our *Mix-Mapped* partitioning model, the vertical part plays a relatively insignificant role in learning procedure; rather it contributes to reducing communication costs by processing the related neurons within the same device. Instead of cloning the entire neurons and putting a whole burden of computational load at the device, we selectively clone neurons by their importance to maintain accuracy without making backup devices overloaded with the additional computational load.

We calculate the importance measure of neurons by taking into account the weight values. We borrow an idea from a previous work [20] that uses the absolute weight value to quantify the strength of an edge connection from one neuron to another. We calculate the importance score with L2-normalization, which used in pruning to rank filters or neurons on the neural network. Then, we pick up the top  $T$  percentage of neurons from malfunctioning devices based on the importance score. The parameter of  $T$  can be determined by the balance between training performance and computational workload.

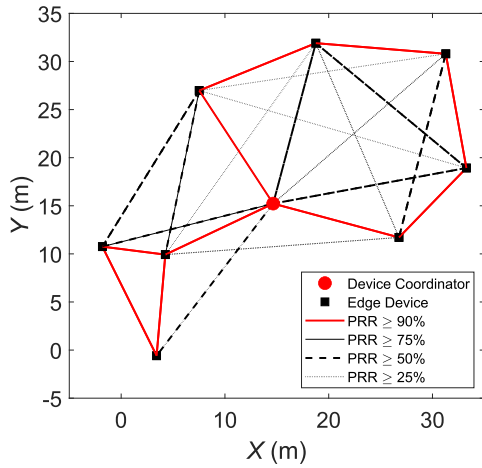


FIGURE 6. Simulated network topology in which one device coordinator and 8 devices are deployed over  $40 \times 40 m^2$ .

Our *neuron cloning* takes place during training and testing procedures. Once the importance score-based cloning procedure is executed, the rest of training proceeds to fine-tune the weight parameters until convergence. The neuron-to-device arrangement is maintained until the end of learning. The device coordinator receives the converged weight values from participating devices and manages the information in the virtual neuron space in preparation for potential link and device failures in the near future.

### V. EVALUATION

We implemented a distributed neural network model from scratch using Python 3.7, in order to keep track of device-wise computation and networking along the forward and backward learning paths. We used the MNIST dataset with the handwritten digits, EMNIST-Letters dataset with the handwritten characters, and the Fashion-MNIST dataset with the images of fashion products to validate our proposed algorithm in a simulated network topology over  $40 \times 40 m^2$ , with an edge network of one device coordinator device and 9 edge devices including device coordinator, as shown in Fig. 6. We mainly used Fashion-MNIST dataset to validate the performance of proposed scheme. To stress out the main model partitioning performance over lossy device-to-device links, the edge devices are assumed to be homogeneous in the computation, memory, and communication resource in the experiments.

The wireless radio communication is simulated with a path-loss shadowing model [23], which captures the core properties of wireless communication. The larger the path-loss exponent (PLE) is, the more volatile the wireless network becomes. A PLE value of 3.9, which is usually used for the indoor environment, is selected, unless otherwise noted. The average PRR is 49.0% where the PRR is measured by counting the delivered packet among 50 packets, within the communication range. With a separate experiment in order to select a target volatile network topology, we have

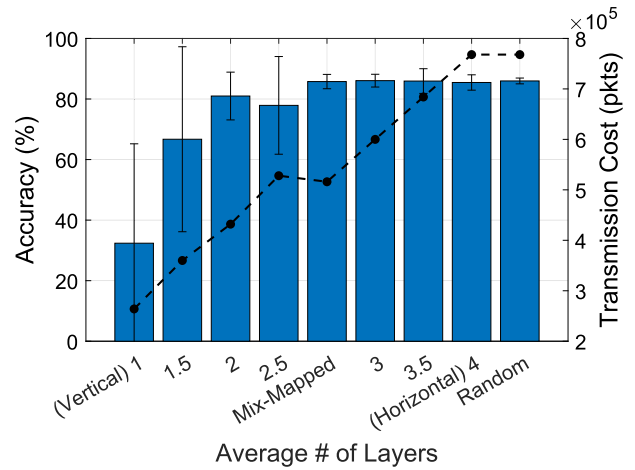


FIGURE 7. Performance of accuracy and communication cost with respect to the number of layers at each submodel with the Fashion-MNIST dataset.

periodically generated packets between wireless links and have taken the average packet reception rate as PRR for a specific link. Assuming 802.11 Wi-Fi interface to be equipped at edge devices, the maximum transmission time of 100 ms between devices is used for quantifying the networking time, as a worst-case network delay.

We use a basic deep neural network (DNN) architecture where the number of hidden layers is chosen to be 4 with a width of 64 across all the layers, which are feasible depth and width at the edge device level. We use Xavier initialization [24] as the weight initializer, Adam optimizer as the optimizer, and ReLU as the activation function. The datasets are split to 60,000 training samples and 10,000 testing samples for the MNIST and Fashion-MNIST datasets, and 88,800 training samples and 14,800 testing samples for EMNIST-Letters dataset. A batch size of 100 is used for all datasets. We ran 10 different runs for each experiment and report the average performance on the test set. It should be noted that both training and testing have been conducted under the link or device failure environments between devices and the device coordinator without using any retransmission or any transport layer support.

### A. EFFECTS OF MIX-MAPPED PARTITIONING

#### 1) ACCURACY AND COMMUNICATION COST

We examine how model partitioning affects training process and communication cost in volatile wireless network environments. We compare with a state-of-the-art model parallelism [4] called *Vertical* partitioning and *Horizontal* partitioning. The *Vertical* makes each device in charge of the neurons from a single layer while *Horizontal* is fairly distributed across layers. We measured broadcast transmission cost and accuracy of the partitioned models in Fig. 7. In Fig. 7, *Vertical* represented by 1 on the x-axis shows very poor performance with only 33%. This implies that assigning all of the neurons at a single layer to very few devices is very vulnerable to the intermittent wireless lossy network environment. This

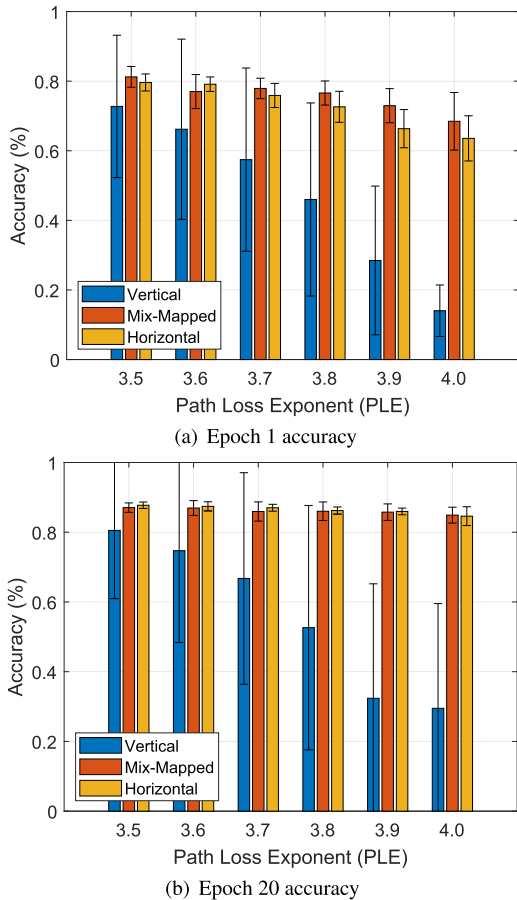


FIGURE 8. Effect of the link failure on the prediction quality as the link volatility gets worse.

is because the *Vertical* neuron assignment critically breaks down the crucial connection along the forward and backward passes in deep neural networks. The accuracy of the model increases from *Vertical* to *Horizontal* represented by 4 and random on the x-axis. Since there are many neurons, a random distribution of the neurons has caused every device to handle some part of neurons at each layer, leading to the average number of layers value of 4. Although the number of neurons of each layer assigned to the device may vary, the *Random* partitioning model shows similar performance with *Horizontal* model.

However, the communication cost increases accordingly because devices in charge of all layers need communication. Therefore, it is necessary to find a balance between accuracy and communication cost. *Mix-Mapped* shows higher and stable accuracy with lower communication cost compare to 2.5 which has same average number of layer. This implies that our *Mix-Mapped* partitioning is a reasonable way to reduce communication costs while stably obtaining high learning accuracy.

We investigate the benefit of our *Mix-Mapped* partitioning varying the PLE in the same network topology. As shown in Fig. 8, as the path-loss exponent increase from 3.5 to 4.0, the

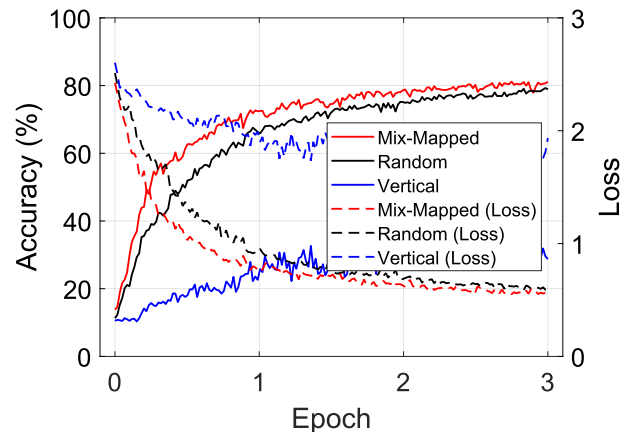


FIGURE 9. Accuracy and loss performance among different partitioning models using Fashion-MNIST dataset.

average PRR value between communicating devices drops from 0.91 down to 0.65. As the wireless network environment gets more lossy, *Mix-Mapped* still holds a high accuracy of 85.76% under the path-loss exponent of 3.9, measured at epoch 20.

*Mix-Mapped* showed higher accuracy at epoch 1 shown in Fig. 8(a) as the connection getting lossy. This result indicates that our partitioning model quickly boots up the learning procedure in volatile network environments. The performance at epoch 20 the accuracy eventually converges to be similar to that of the random model as shown in Fig. 8(b), but consumes lower communication cost compared to the random model. *Mix-Mapped* can decrease the convergence speed of learning with reasonable communication cost.

We compare the *Mix-Mapped* partitioning with the together with *Random* partitioning and *Vertical*, while fixing the random seed. As shown in Fig. 9, as the training epoch goes on, both algorithms get improved the accuracy on the test set, while their losses on the training set drops. *Mix-Mapped* partitioning works better than the *Random* and *Vertical* partitioning algorithm.

In order to see how effective the device mapping is in the *Mix-Mapped* model, we compared our mapping method with randomly allocated (*Mix-Mapped (Random Mapping)*) and reversed order (*Mix-Mapped (Reverse Mapping)*) allocation, where the devices with high connectivity are vertically assigned, and the devices with low connectivity are horizontally assigned. Our mapping strategy shows the highest accuracy over training epochs in Fig. 10. It indicates that assigning horizontal portion to device with high connectivity achieves reliable performance. This is because the forward and backward propagation in the horizontal part of the model pass by forming strong learning paths with a high probability.

## 2) DEVICE COORDINATOR AND PATH CONSTRUCTION

We examine how device coordinator affect training accuracy. Fig. 11 shows that our calculation based selected device

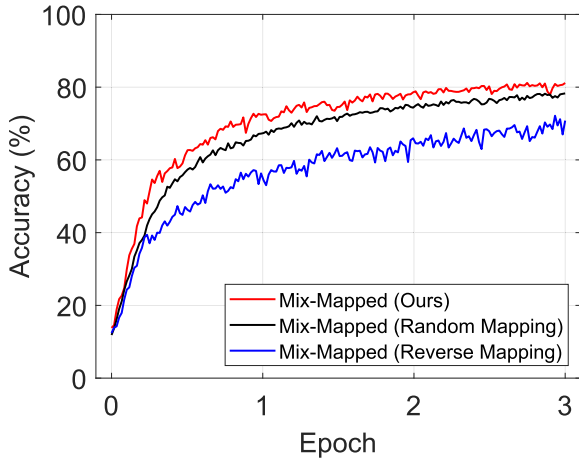


FIGURE 10. Performance of accuracy with different device allocation on the Fashion-MNIST dataset.

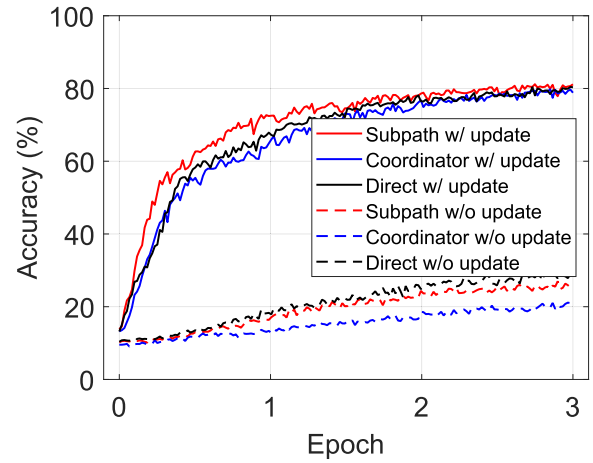


FIGURE 12. Comparison of partial updates and no-updates on Fashion-MNIST dataset.

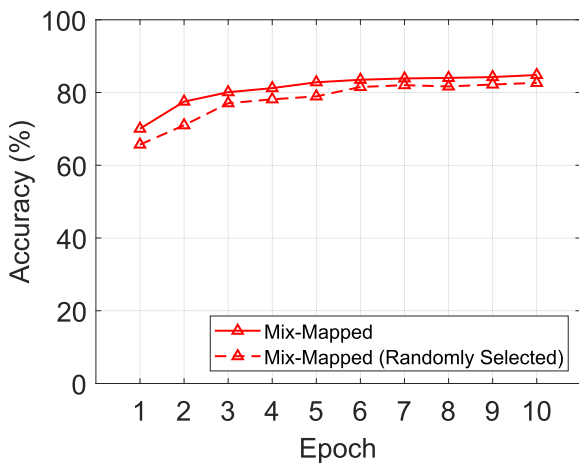


FIGURE 11. Accuracy performance of calculated Device Coordinator and randomly selected Device Coordinator, with respect to epoch.

coordinator (*Mix-Mapped*) shows better performance than the one selected randomly. This result indicates that if the device coordinator is effectively selected, the coordinator serves to aggregate and send results between layers, increasing the probability that the device receives results.

We also investigated the effect of partial update, by zero padding the failed delivery values, or use its own previous layer value as an input. In the Fig. 12, the result is not calculated if the result is not received from previous layers, and accordingly, the result of the continuous layer is not calculated. Updating partial results is crucial. Moreover, when using subpath, it shows fast convergence and better performance relative to using path that goes through *Device Coordinator*, when there is and is no partial update. When comparing results of *Subpath w/ update* and *Subpath w/o update*, we can feel the using partial update is very crucial, and learning can not happen when there is no partial update in lossy environment.

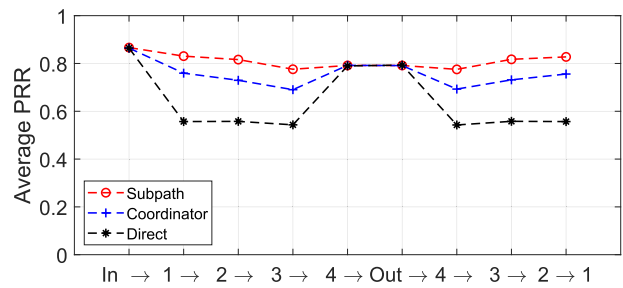


FIGURE 13. Average PRR value during full forward and backward pass.

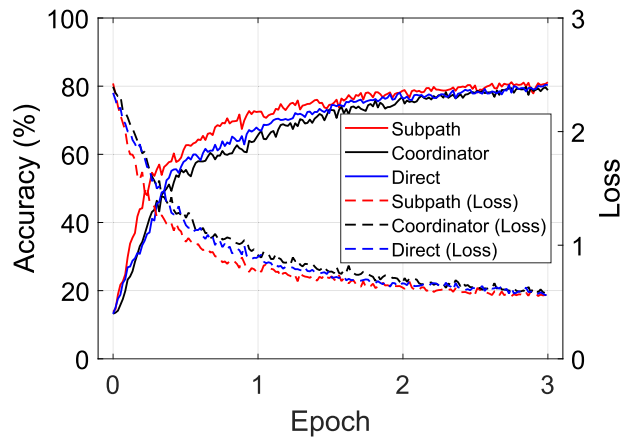


FIGURE 14. Dynamic accuracy and loss performance of *Mix-Mapped* with path of direct, *Device Coordinator*, and *subpath* communication with respect to epoch.

To validate the effect of subpath construction, we investigate the case where there is no subpath using only the device coordinator represented as *Coordinator* in Fig. 14 and Fig. 13, and the case of using only the direct communication (*Direct*) without using the device coordinator. As shown in Fig. 13, the communication success rate of the device to the consecutive layer is highest when using the subpath. The average PRR values for *Subpath* for every



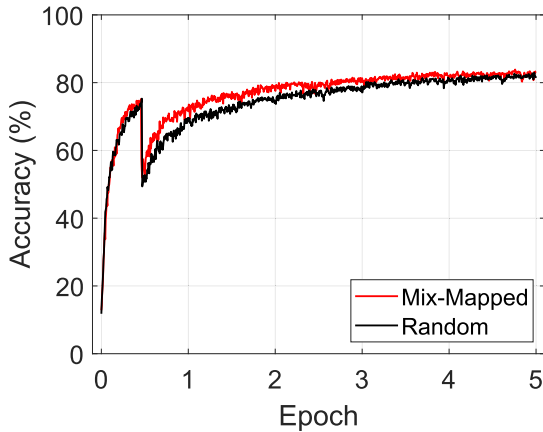


FIGURE 15. Accuracy over epochs when PLE value is changed from 3.7 to 3.9 at accuracy of 0.75.

layers is 0.81, while *Direct* shows only 0.64. By utilizing both the direct communication and the coordinator, the result can be sent through the coordinator when direct communication is volatile, allowing the devices to receive and update results more frequently increasing accuracy in Fig. 14. The device also capable of direct communication to other devices not through the coordinator, so communication costs can be saved.

**B. MODEL RECOVERY**

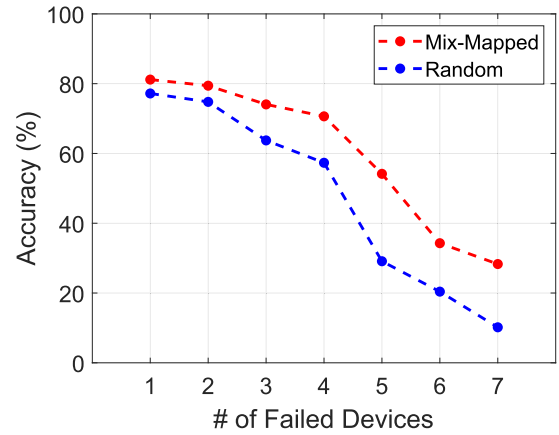
To observe how the *Mix-Mapped* model recovers, we run two cases: when the network environment was changed during the training and when during the testing.

1) LINK DYNAMICS DURING TRAINING

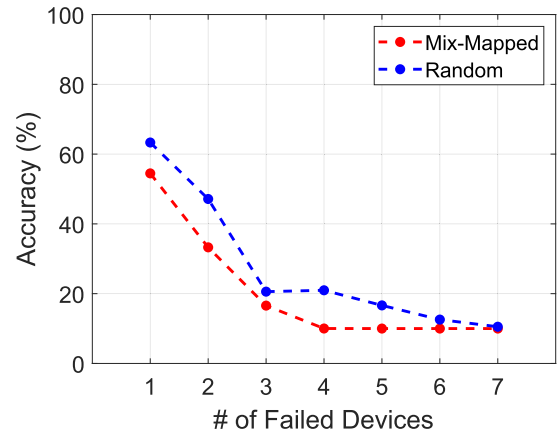
We intentionally changed in a network environment with a PLE from 3.5 to 3.9 when the accuracy of training models reached 0.75 during training. As shown in Fig. 15, *Mix-Mapped* model recovers faster than the *Random* partitioned model, and it converges eventually approximately as the training progresses. Therefore, the *Mix-Mapped* model cope with frequently changing wireless communication characteristics which can help to progress the learning faster.

2) EFFECT OF SEVERE DEVICE FAILURES DURING TESTING

To examine the effect of severe device failures, we make a device one-by-one inactive with the order from the weakest to the strongest connectivity to the device coordinator and vice versa, at testing phase. The neuron cloning is applied for both partitioned models as shown in Fig. 16(a) and Fig. 16(b). It should be noted that both models did not reflect the device failure situation, and the performance is measured during the test phase after the failure-free training phase is over. Each partitioned model is trained with 9 devices including a device coordinator, and the test accuracy performance is measured with respect to the number of failed devices. In Fig. 16, when a device with the weakest connectivity was first removed, some parts of neurons from the affected device are cloned



(a) Model failure from the worst connected device



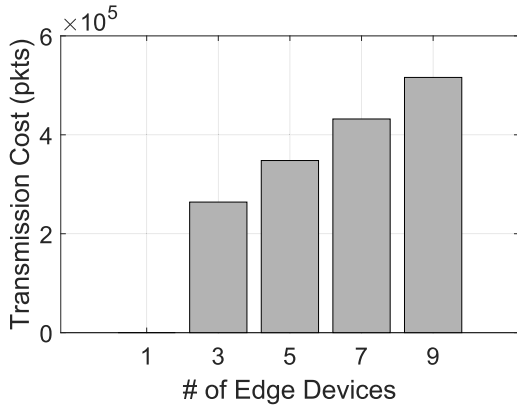
(b) Model failure from the best connected device

FIGURE 16. Performance with respect to the number of failed devices over the failure severity level, on the Fashion-MNIST dataset.

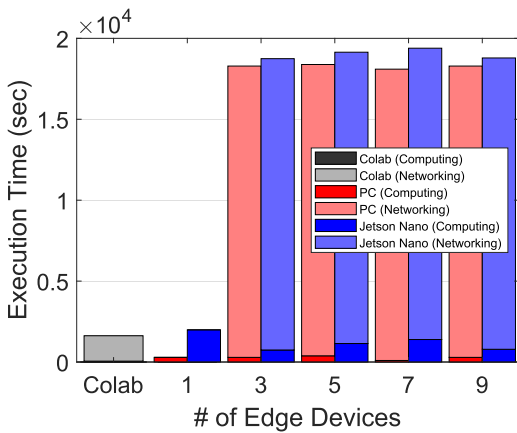
based on the priority. The parameter of  $T = 25$  is used, where top 25% of malfunctioning neurons are cloned. In Fig. 16, our *Mix-Mapped* shows better performance than *Random* since our *Mix-Mapped* structure provides devices to hold more neurons from one layer since the devices with weak connectivity are aligned in a vertical manner. For this reason, the horizontal device can smoothly deliver many neurons to next layer within itself and has stably connected with devices. On the other hand, *Random* only get some portion of layers since they are all aligned in a semi-horizontal shape. If there are only one device left, the performance of *Mix-Mapped* is 28.30%, whereas *Random* reached at 10.17%, resulting in almost 20% gap between two models. In Fig. 16(b), although our *Mix-Mapped* appears to have a lower performance than *Random*, the gap between two models is not noticeable, and the maximum gap is only 10.96% in case of the four failed device scenario.

**C. NETWORKING AND COMPUTATION OVERHEAD**

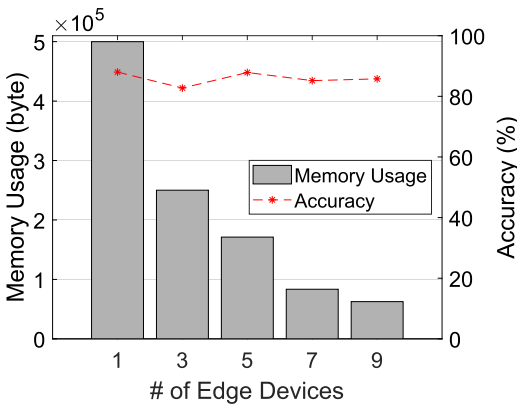
We measure transmission cost in *StitchNet* for sharing the intermediate computation results between devices in terms of the number of participating edge devices. As a default experiment setting, 9 devices including a device coordinator are used in Fig. 17. We quantify the number of transmitted



(a) Transmission cost in terms of packets as the number of available devices increases



(b) Computation and networking overhead comparison in time complexity over different platforms: Google Colab, desktop PC, and Jetson Nano



(c) Performance dynamics in terms of per-device memory usage and accuracy in terms of the number of participating edge devices involved in the computation

**FIGURE 17. Networking and computation overhead when the number of devices varies using the Fashion-MNIST dataset.**

packets, as in Fig. 17(a). As more edge devices participate in a distributed learning model, the number of transmitted packets increases accordingly due to the fact that the computation capacity is shared by the network overhead among devices. This shows an interesting trade-off between computation and networking.

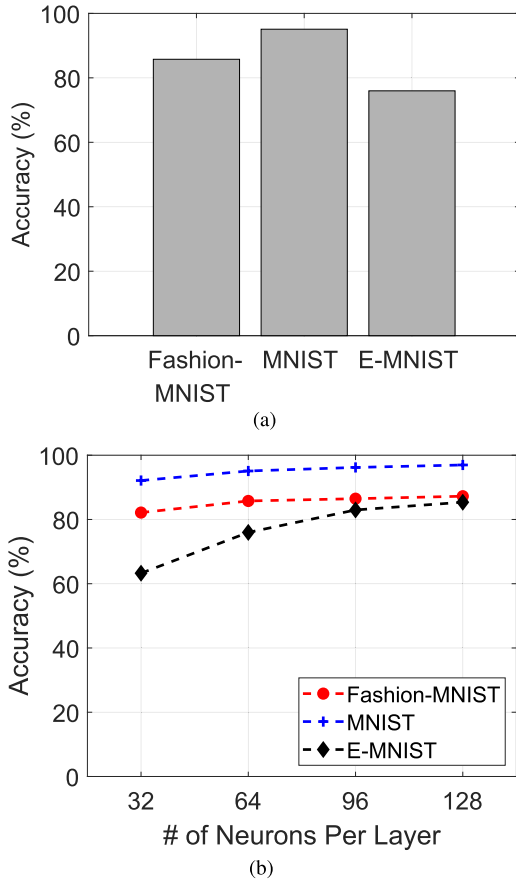
We also quantify both networking and computation overhead in time, by running *StitchNet* over different platforms: Google Colab with GPU, desktop PC with Intel Core i7-6700 CPU and 8 GB RAM, and NVIDIA Jetson Nano 4GB, as shown in Fig. 17(b). By constructing a set of models with each different weight size at a device, we analytically measure the running time at different devices under the same experimental condition. It is an expected result that a Google Colab cloud server with a strong computation power takes only a few hundreds of milliseconds for computation, whereas the data upload takes even longer than the computation time instead.

If only the edge devices, such as Jetson Nano are used, our *StitchNet* makes a deep learning model still feasible even at the computation and memory constrained devices. If only one device runs the model by processing all of the neurons, e.g., at the device coordinator itself, it takes huge amount of time as expected. However, as more devices become available for distributed learning, the neurons can be distributed over more devices, making the computation job quickly finished. In return, the distributed model needs to share the intermediate computation results among devices for collaborative learning, increasing the communication overhead.

Further, we investigate the effect of different number of edge devices on performance in terms of per-device memory usage and accuracy. We measure the memory usage at a single device by using the Python system function. When only one edge device is used, the device coordinator takes over the whole computation, and thus, there is no network communication or network failure, which can be considered as the ideal case. In Fig. 17(c), it is obvious that the per-device memory usage decreases as more edge devices are involved in the computation, while the accuracy has sustained with only 2.24% dropped, compared to the ideal single device case. This result implies that our *StitchNet* offers a stable learning structure ensuring sufficient performance with smaller memory usage at the edge, by making even wirelessly connected computation resources *stitched together* in an efficient manner.

**D. PERFORMANCE IN DIFFERENT DATASETS**

We investigated how our *Mix-Mapped* scheme performs across different datasets, using Fashion-MNIST, MNIST, and E-MNIST, with respect to different size of the deep learning model in Fig. 18. In Fig. 18(a), we quantified the accuracy using the model size of 64 neurons, while being trained for 20 epochs and tested 100 times for the same model. The accuracy on the MNIST dataset was the highest since it requires relatively easier tasks than the other datasets for a model to serve. Using the E-MNIST dataset, our model showed the lowest performance since it consists of harder tasks with 26 categories. In Fig. 18(b), as the number of neurons increases, the overall accuracy improves up to 96% on MNIST, 87% on Fashion-MNIST, and 85% on E-MNIST using a model size of 128 neurons. This result indicates



**FIGURE 18.** Accuracy over different dataset and then number of neurons per layer.

that our *Mix-Mapped* scheme with learning path construction offers generally stable performance even under wireless lossy links.

## VI. CONCLUSION

We have presented a new (semi-) distributed learning framework that can operate under volatile wireless network environments. By exploiting some key components of *device cohesion* and *neuron cloning*, our proposed *StitchNet* architecture makes a distributed deep learning feasible at resource-constrained edge devices, while achieving high prediction quality with resilience to dynamic changes.

For future work, it would be interesting to devise a device coordinator-less model partitioning method that can locally aggregate and distribute partial information, considering heterogeneous edge devices with different memory and computation capacity. Adapting the underlying architecture, such as convolutional neural network with network volatility for a different application, (e.g., natural language processing), would be another future direction. Moreover, we can consider an edge-based hybrid parallelism that combines both data and model parallelism that can relieve some possible privacy issues such as packet sniffing towards more distributed edge learning.

## REFERENCES

- [1] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 4700–4708.
- [2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
- [3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, vol. 1, Jun. 2019, pp. 4171–4186.
- [4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, and J. Klingner, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*.
- [5] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
- [6] Y. Bengio, "Deep learning of representations: Looking forward," in *Proc. Int. Conf. Stat. Lang. Speech Process.* Cham, Switzerland: Springer, 2013, pp. 1–37.
- [7] R. Mayer and H.-A. Jacobsen, "Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools," *ACM Comput. Surveys*, vol. 53, no. 1, pp. 1–37, Jan. 2021.
- [8] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, B. McMahan, and T. Van Overveldt, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst.*, vol. 1, 2019, pp. 374–388.
- [9] W. Y. B. Lim, N. C. Luong, D. T. Hoang, Y. Jiao, Y.-C. Liang, Q. Yang, D. Niyato, and C. Miao, "Federated learning in mobile edge networks: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 3, pp. 2031–2063, Apr. 2020.
- [10] H. Brendan McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. Y. Arcas, "Communication-efficient learning of deep networks from decentralized data," 2016, *arXiv:1602.05629*.
- [11] D. Bahdanau, K. H. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *Proc. 3rd Int. Conf. Learn. Represent.*, 2015, pp. 1–15.
- [12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, and M. Ranzato, "Large scale distributed deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1223–1231.
- [13] 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 *Proc. 27th ACM Symp. Operating Syst. Princ.*, Oct. 2019, pp. 1–15.
- [14] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, and Y. Wu, "GPipe: Efficient training of giant neural networks using pipeline parallelism," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 103–112.
- [15] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *Proc. 22nd Int. Conf. Architectural Support Program. Lang. Operating Syst.*, New York, NY, USA, Apr. 2017, pp. 615–629, doi: [10.1145/3037697.3037698](https://doi.org/10.1145/3037697.3037698).
- [16] S. Teerapittayanon, B. McDanel, and H. T. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 328–339.
- [17] V. Turina, Z. Zhang, F. Esposito, and I. Matta, "Federated or split? A performance and privacy analysis of hybrid split and federated learning architectures," in *Proc. IEEE 14th Int. Conf. Cloud Comput. (CLOUD)*, Sep. 2021, pp. 250–260.
- [18] A. Li, J. Sun, X. Zeng, M. Zhang, H. Li, and Y. Chen, "FedMask: Joint computation and communication-efficient personalized federated learning via heterogeneous masking," in *Proc. 19th ACM Conf. Embedded Networked Sensor Syst.*, Nov. 2021, pp. 42–55.
- [19] O. Gupta and R. Raskar, "Distributed learning of deep neural network over multiple agents," *J. Netw. Comput. Appl.*, vol. 116, pp. 1–8, Aug. 2018.
- [20] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.

- [21] A. Cerpa, J. L. Wong, M. Potkonjak, and D. Estrin, "Temporal properties of low power wireless links: Modeling and implications on multi-hop routing," in *Proc. 6th ACM Int. Symp. Mobile Ad Hoc Netw. Comput.*, 2005, pp. 414–425.
- [22] M. M. Afsar and M.-H. Tayarani-N, "Clustering in sensor networks: A literature survey," *J. Netw. Comput. Appl.*, vol. 46, pp. 198–226, Nov. 2014.
- [23] A. Goldsmith, *Wireless Communications*. Cambridge, U.K.: Cambridge Univ. Press, 2005.
- [24] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, 2010, pp. 249–256.



**JIHO LEE** is currently pursuing the Ph.D. degree in computer science and engineering with the University of Virginia, Charlottesville, VA, USA. Her current research interests include the IoT, sensor networks, wireless *ad-hoc* networks, and deep learning.



**JEIHEE CHO** is currently pursuing the M.S. degree in computer science and engineering with Ewha Womans University, Seoul, South Korea. Her current research interests include the IoT, sensor networks, wireless *ad-hoc* networks, and deep learning.



**HYUNGJUNE LEE** (Member, IEEE) received the B.S. degree in electrical engineering from Seoul National University, Seoul, South Korea, in 2001, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, USA, in 2006 and 2010, respectively. He joined Broadcom as a Senior Staff Scientist for working on research and development of 60GHz 802.11ad SoC MAC. Also, he worked at the AT&T Laboratories as a Principal Member of Technical Staff with the involvement of LTE overload estimation, LTE-WiFi interworking, and heterogeneous networks. He is currently an Associate Professor with the Computer Science and Engineering Department, Ewha Womans University. His current research interests include distributed learning and future wireless networks on the IoT, fog computing, VANET, and machine learning-driven network system design.

• • •