NeuroMessenger: Towards Error Tolerant Distributed Machine Learning Over Edge Networks

Song Wang, Xinyu Zhang
University of California San Diego
{sowang, xyzhang}@ucsd.edu

***Abstract*—Despite the evolution of distributed machine learning (ML) systems in recent years, the communication overhead induced by their data transfers remains a major issue that hampers the efficiency of such systems, especially in edge networks with poor wireless link conditions. In this paper, we propose to explore a new paradigm of error-tolerant distribute ML to mitigate the communication overhead. Unlike generic network traffic, ML data exhibits an intrinsic error-tolerant capability which helps the model yield fair performance even with errors in the data transfers. We first characterize the error tolerance capability of state-of-art distributed ML frameworks. Based on the observations, we propose NeuroMessenger, a lightweight mechanism that can be built into the cellular network stack, which can enhance and utilize the error tolerance in ML data to reduce communication overhead. NeuroMessenger does not require per-model profiling and is transparent to application layer, which simplifies the development and deployment. Our experiments on a 5G simulation framework demonstrate that NeuroMessenger reduces the end-to-end latency by up to 99% while maintaining low accuracy loss under various link conditions.**

I. Introduction

Owing to the standardization of 5G multi-access edge computing (MEC) [1] and booming development of machine learning (ML) applications, we have witnessed a growing interest in synergizing the user equipment (UE), MEC, and the cloud to boost mobile ML in recent years [2]–[4]. The combined computation power from many devices and the flexibility of splitting workloads enable this new distributed ML paradigm to break the limits of scarce computation resource and unpredictable latency, which used to hinder the UE and cloud based execution, respectively. Among the many proposed solutions of distributed ML, two have recently gained major traction: *Split ML* [5]–[8] dynamically assigns parts of a model *inference* process to different computing nodes based on network conditions and computation resources, to alleviate the pressure of computation on UE devices and potentially optimize end-to-end latency and energy consumption [3], [5], [8]–[10]. *Federated Learning (FL)*, on the other hand, distributes the *training* of a model to a federation of participant devices. Each device (usually a UE) trains a copy of the model with its local data, and updates the learned weights to a central parameter server (PS) for aggregation. An FL system utilizes the computation power of a large number of mobile devices, while eliminating the need for uploading privacy-sensitive raw data to the cloud [11], [12].

Despite the unprecedented computation power scale-up, these distributed ML methods suffer from one major bottleneck: due to their distributed architecture, computation nodes need to exchange data with each other (referred to as *ML data*) intensively, which may cause significant communication overhead. Specifically, *split ML transmits intermediate layers' output data from the UEs to MECs, and FL's participating devices upload the locally trained model weights to the parameter server*. Since the data source is usually located at the UE, split ML and FL mainly rely on the cellular uplink for data transfer. Due to the limited power budget of mobile devices, the uplink typically experiences lower link quality [13]. As illustrated in Fig. 1, under poor link conditions, block errors occur more frequently and the cellular link needs to attempt multiple retransmissions to correct the errors. This tends to hamper latency sensitive ML applications such as remote driving, remote-controlled robotics, and AR display/gaming which require a stringent end-to-end latency of around 10 ms [14].

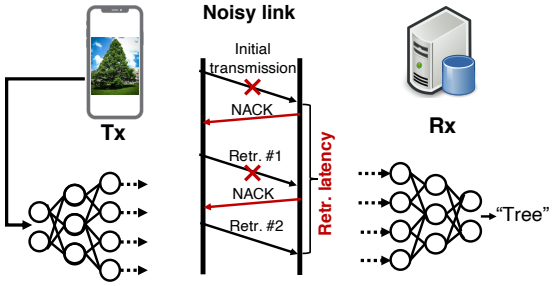

Fig. 1: An example of split ML over a lossy edge network.

Existing works have approached the communication overhead problem by compressing the ML data before transmission [15], [16]. Such approaches often need non-trivial redesign on the application level, and are unaware of the low-layer overhead due to poor link quality. In this paper, we propose to leverage the intrinsic error-tolerant capability of the ML data to circumvent the communication overhead. Unlike the general cellular network traffic that requires error-free data transmission, we observe that distributed ML may still perform the inference or training operations with fair accuracy, even with the present of errors in data transfer. As a result, retransmission becomes unnecessary as long as the error-induced accuracy loss is tolerable. In other words, such error tolerance capability in ML data reduces the need for retransmissions and consequently, the overall communication overhead.

To realize this principle, we present a first study on the error-tolerant capability of distributed ML models. By examining representative split ML and FL model execution over 5G edge networks, we address the following key questions:

How well can the ML data tolerate errors? We perform a layer-level characterization of the accuracy performance versus

error rate under a wide range of link conditions and block error patterns. Our experiments reveal that the error tolerance capability exists in most of the commonly used deep neural network (DNN) layers in split ML and FL. However, the specific accuracy losses vary significantly across different types of DNN layers. For a certain layer, the accuracy loss and error rate are strongly correlated, and hence the accuracy performance of the ML model can be profiled as a function of error rate. In addition, the ML data in split ML suffer more accuracy loss from burst errors, whereas FL is agnostic to error patterns.

How to enhance the error-tolerant capability of ML models? Based on the error-tolerance characterization, we propose two techniques to enhance error tolerance in distributed ML: *Interleaved coding* utilizes the property that split ML data is prone to burst errors. It randomly interleaves the ML data sequence so that the burst errors are jumbled into smaller pieces and spread over time, which mitigates their impacts on the model accuracy. *Importance-based coding* facilitates Unequal Error Protection (UEP) on parameters with different values and reduces the error rate on high-valued parameters which are more likely to impact the model accuracy.

How to leverage the error-tolerant capability to improve the efficiency of distributed ML in wireless edge networks? We propose NeuroMessenger, a lightweight cellular-native mechanism that reduces the latency of distributed ML over edge networks. NeuroMessenger is jointly executed by the transmitter and receiver of the ML data, *i.e.* UE and basestation co-located with MEC. It is transparent to ML applications and tightly integrated with the devices' protocols stack which aligns with 3GPP's vision for future edge intelligence system [14]. For an ML model, NeuroMessenger performs a layer-wise profiling of the error rate to accuracy mapping offline. At runtime, NeuroMessenger on transmitter first reduces the ML data size by pruning the redundant parameters. Then it enhances the error tolerance of the ML data by applying the aforementioned enhancement schemes. The encoded data is then sent over the cellular edge link. At the receiver end, NeuroMessenger's retransmission controller retrieves an estimation of the channel state and estimates the corresponding error rate. Then, based on the error rate to accuracy profiles from the offline stage, the retransmission controller predicts if the accuracy can meet the application requirement and determines if a retransmission is necessary and how aggressive the retransmission should be. By shrinking the need for retransmission, NeuroMessenger substantially improves the communication efficiency of distributed ML while maintaining a user-defined accuracy requirement. With NeuroMessenger, the edge link can aggressively choose a high order modulation and coding scheme, which leads to high raw bit-rate and high block error rate under moderate or low channel quality. NeuroMessenger also greatly reduces the complexity of distributed ML system's development and deployment by making the communication overhead reduction mechanism transparent to applications.

We evaluate NeuroMessenger on a 5G NR simulator. Our experiments adopt the typical PHY settings of a 5G uplink (*e.g.*, modulation and coding scheme, bandwidth and sub-carrier spacing, transport block size, *etc.*). We choose state-of-art image classification and speech recognition models as representative ML applications. Our experiment results show: (i) NeuroMessenger reduces the communication latency by up to 95% comparing to baselines while still maintaining less than 10% accuracy loss under an extreme link SNR of -2 dB. (ii) NeuroMessenger is effective under a wide range of error rates. We observe 20% to 99% latency reduction and less than 5% accuracy loss under 0.1 to 0.95 block error rate. (iii) For split ML models, NeuroMessenger is effective regardless of the partitioning point within the models.

The major contributions of this paper are as follow: (i) A First characterization of error tolerant capability in distributed ML. (ii) A novel system that enhances and utilizes error tolerant capability to reduce communication overhead in distributed ML. (iii) Experimental verification of NeuroMessenger on a 5G edge network environment with representative distributed ML settings.

II. Related Work

A. Distributed Edge Intelligence

Distributed ML on cellular edge servers has garnered much interest in the past two years. In particular, the 3GPP standardization group has envisioned a deep integration of split ML and FL for mobile inference and training respectively [17]. Existing research on split ML mostly focused on partitioning a DNN's computing load to two or more parts which are executed by generic edge servers to meet latency or energy optimization objectives [5]–[7]. The research on FL proposed new designs on aggregation algorithm [12], [18], [19], participants selection [20]–[22], and incentive mechanisms [23], [24] to improve the communication efficiency or privacy. These works assume the ML data is transmitted directly over the communication links which usually over-protect the data integrity. When the link condition is poor, frequent retransmissions may occur. *Even under good link conditions*, the communication PHY layer conservatively chooses the modulation and coding scheme (MCS) that is most likely to work in an error-free manner, rather than choosing one with high raw bit-rate but more block errors. In contrast, we propose to build error tolerant capability into the intermediate data, so that they can be salvaged in spite of errors. With this measure, the edge network can avoid the costly retransmissions, and can aggressively choose a high but error-prone MCS level to improve communication efficiency.

B. ML communication overhead reduction

A classical approach to reduce the communication overhead in edge ML is to compress the ML data. Yao *et al.* [25] proposed a NN-based compressor/decompressor design that compresses the intermediate data following compressive sensing theory. Hu *et al.* [26] adopted a similar NN-based compressor/decompressor and leveraged the prior transmissions to aid the decompression of current intermediate data. These

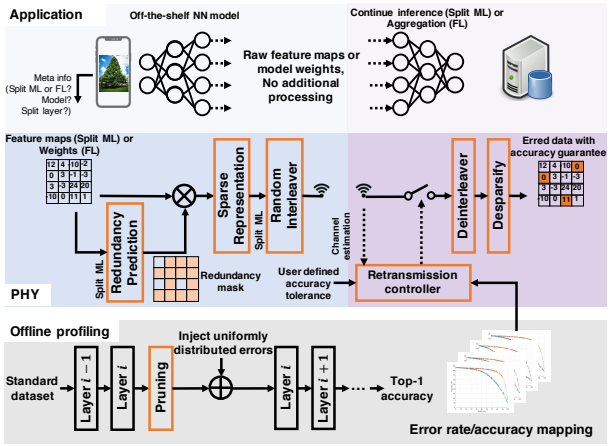

Fig. 2: NeuroMessenger system overview.

NN-based compression techniques greatly reduce the intermediate data size, but they bear two limitations. First, the NN compressor/decompressor themselves require hours of training for individual model and split point [25]. In the case where the split point or the ML model needs to be constantly changed due to network dynamics, *e.g.*, in the context of online learning, it is impossible to train the compressor/decompressor in real time. Second, such compression approaches are heavily customized to individual application, *e.g.*, the compressor/decompressor are jointly trained with the ML model. In contrast, NeuroMessenger only needs a lightweight application-independent profiling consisting of only the inference process which usually takes less than $1/1000$ of time to run compared to the training.

III. System Overview

The system architecture of NeuroMessenger is shown in Fig. 2. At runtime, the UE application runs off-the-shelf distributed models. It can pass the raw ML data to low layers along with a small metadata indicating if the application is split ML or FL, what the NN model is, and the splitting point for split ML. NeuroMessenger, as part of the cellular-native stack, recognizes the metadata and acts on the ML data accordingly. For split ML, NeuroMessenger first prunes the redundant data (Sec. V-B), and then applies error tolerance coding (Sec. V-A). As for FL, NeuroMessenger skips these two processes. The resulting data and metadata are then fed to the original cellular PHY layer and then transmitted to the basestation.

Upon receiving the data, the NeuroMessenger retransmission controller on the basestation estimates how many retransmission attempts are needed to balance the model accuracy and communication overhead. Specifically, the retransmission controller first uses the information in the metadata to find the corresponding error rate to accuracy mapping function G, which is profiled offline (Sec. V-B). It also retrieves the latest uplink SNR measurement and estimates the corresponding error rate r. Then, the controller compares estimated accuracy under the current error rate $G(r)$ and a user-defined minimal tolerable accuracy a_u and calculates the maximum number of retransmission attempts n as follow:

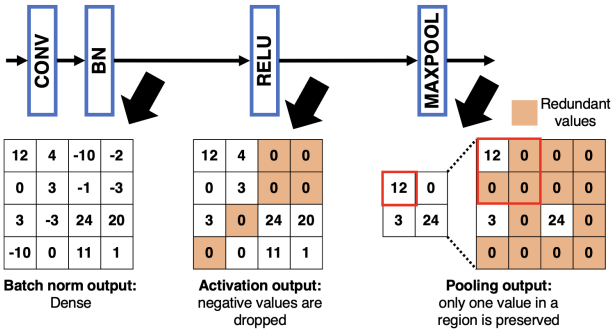

Fig. 3: Layer composition of a typical neural network and a demonstration of feature map redundancy: most of parameters in the feature map from batch norm layer are dropped after the pooling layer.

$$n = \begin{cases} 0 & \text{if } G(r) > a_u \\ F(r, a_u) & \text{if } G(r) < a_u \end{cases} \quad (1)$$

where $F(r_1, r_2)$ calculates the minimal number of retransmission attempts to reduce error rate from r_1 to r_2. For example, $F(0.8, 0.5) = 2$ means that to reduce error rate from 0.8 to 0.5, the maximal number of retransmission attempts needed is 2. F is determined by the PHY layer MCS level and the channel condition, and can be derived by existing predictive models [27]. The basestation then signals the UE for retransmission until n attempts or the data passes parity check. By eliminating the retransmissions when $G(r) > a_u$ and reducing the number of retransmission attempts when $G(r) < a_u$, NeuroMessenger alleviates the communication overhead.

Notably, NeuroMessenger separates the ML data transfer from the main ML design, and thus it reduces the complexity of the distributed ML system design while still providing model-specific efficiency improvement. The lightweight offline profiling also enables easy and fast adaptations of rapidly evolving ML models to NeuroMessenger without any runtime overhead.

IV. Error-tolerance in Distributed ML

In this section, we first characterize the error tolerance capability in a typical split ML setting. Then, based on the characteristics, we propose and verify two techniques to encode the intermediate data transfer and enhance error tolerance in generalized split ML models.

A. A dissection of neural network models

To study the error tolerance in split ML, we must first understand the structures of split ML and the neural network (NN) models. A NN model consists of multiple consecutive layers. During an inference task, a layer takes the output data from its previous layer, commonly referred to as *feature maps*, applies certain operations and feeds its own output feature maps to the next layer. Despite numerous variants, most commonly used convolutional neural network (CNN) models share the same 4-layer building blocks, as shown in Fig. 3.

Convolution (conv) layer convolves the input data or feature maps with a set of learned filters.

Batch norm (bn) layer normalizes a batch of feature maps.

Pooling (maxpool, avgpool) layer applies the maximum or average function over a region in the feature maps and reduces the region of parameters to a scalar.

Activation (sig, ReLu) layer applies a non-linear activation function to individual parameters and maps the negative or small valued parameters to zero.

In addition to the 4 basic layers in CNN, **fully connected layers**, which linearly combine all feature maps, are often used as a final classifier or an independent NN model for classification tasks.

B. Characterizing error tolerance in split ML

We first demonstrate the impact of errors of intermediate data transfer in split ML, with a pre-trained ResNet18 model [28]. ResNet18 is a popular image classification CNN model consisting of 54 layers. Due to its broad application and typical layer structure, it is often used as the benchmark for distributed ML designs [5], [25]. We assume an example split ML setting where a UE and a MEC split the model at the 5th layer over a noisy 5G link. The parameters in the feature maps are stored in standard 32-byte float type. To investigate the impact of different error patterns, we use two types of errors: (1) *burst errors* represents the common error pattern in packetized communications where an error corrupts an entire packet. To match our 5G link assumption, we set the length of a packet to 3777-byte blocks, a typical transport block size in 5G NR [29]. (2) *random errors* represents a more general but rarer error pattern where an error corrupts an random individual parameter in the feature maps. The corrupted parameters are treated as zeros in our characterization experiments. We vary the error rate and test the corresponding top-1 inference accuracy on the CIFAR100 dataset. Fig. 4 showcases the error rate to accuracy mapping when splitting at two example layers (layer 4 and 26). By examining the error rate to accuracy mapping at each layer, we have the following key observations:

The vanilla split ML has limited error tolerant capability. As shown in Fig. 4, the inference accuracy gradually decreases to 0 as the error rate increases. This implies the feature maps inherently possess error tolerance and may still produce correct inference result under a non-zero error rate. We have the same observation for the error rate to accuracy mappings of all layers except for the fully connected layers. This inherent error tolerant capability is mainly caused by convolution and pooling layers after the splitting point which operate on local regions and often have "soft outputs", *i.e.* float point type. Even if a parameter is corrupted, the convolution and pooling operations in the layers after the splitting point can still produce similar values from other parameters in the same region and eventually dilute the impact of the corrupted parameter. Note that in both figures of Fig. 4, *the accuracy under burst errors is approximately linear with respect to the error rate*, with 7% accuracy loss for every 10% error rate increase. For a slightly noisy 5G uplink with 10% error rate [30], it will lead to 7% accuracy loss, which is usually the accuracy gap between an advanced NN model and a simple classifier. This implies

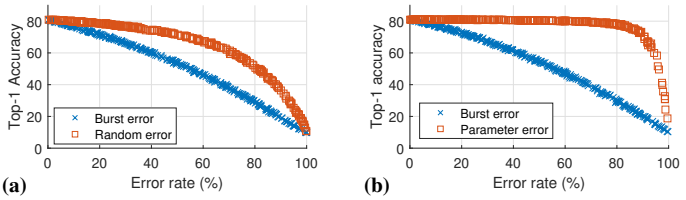

Fig. 4: Top-1 accuracy with different error rate applied to the feature map from (a) layer 4, (b) layer 26.

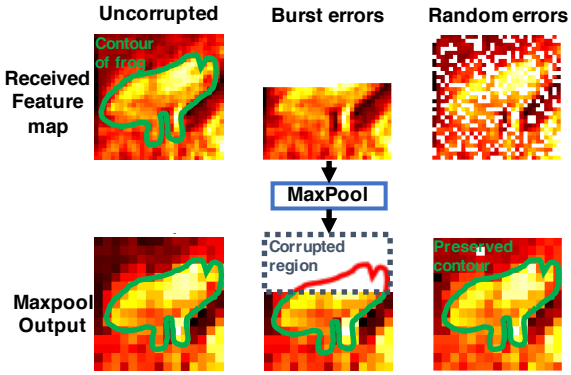

Fig. 5: A demonstration of impact of different types of errors on feature maps: the top region of the frog shape is almost entirely corrupted by block errors, while the random error preserves the shape.

that *the inherent error tolerance in ML data alone can hardly combat the block errors in practical communication systems without a major impact on inference accuracy*.

Error tolerance is layer-dependent. The error rate to accuracy mapping curve differs across layers, indicating that under the same link condition, splitting at different layers yields different accuracy. For example, at block error rate of 0.8, layer 26 and 4 show accuracy of 78% and 52%, respectively. Such gaps are observed for all other layers with accuracy differences ranging from $< 1\%$ to 50%. This seemingly obvious observation reveals the necessity for a layer-wise error rate to accuracy profiling in the cases where the splitting layer is dynamically selected.

Split ML is prone to burst errors. In Fig. 4, we see with the same error rate, the block errors cause far more accuracy loss than the random errors, *i.e.*, up to 20% for layer 4 and 50% for layer 26. For other layers in ResNet18, we also observe similar accuracy gaps ranging from 20% to 60%. Similar to the first observation, such vulnerability to burst errors originates from the convolution and pooling operations. Since these operations are applied to local regions, the output feature maps usually show a similar pattern to the input feature maps in spite of their different sizes. Fig. 5 shows a visualization of input and output feature maps at the 5th layer (maxpooling) of ResNet18 with an image of a frog as input. We see that the basic shape and the contour of the frog are preserved after the maxpooling layer, even though the feature map size reduces from 24×24

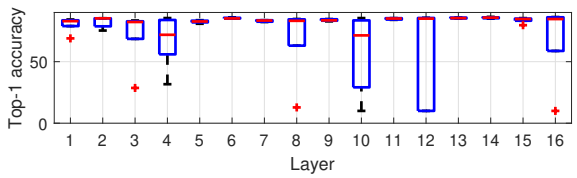

Fig. 6: Top-1 accuracy distribution of VGG11 FL under BLER=0.1. We see the ranges of top-1 accuracy of batch norm layers (3, 4, 8, 10, 12, 16) exceed 20 percent, indicating that the batch norm layers' weights are highly sensitive to errors and the resulting accuracy cannot be estimated by error rate.

to 12×12. As a result, the *random errors in the input feature maps are usually diluted in the following layers*.

Burst errors, on the other hand, corrupt long sequences of parameters that usually span multiple local regions or even an entire input feature map. Since convolution and pooling layers can only produce zeros if all parameters in a local region are corrupted (treated as zero in our experimental characterization), the impact of burst errors is often preserved on the following layers. Fig. 5 showcases such phenomenon. We see that under burst errors, the top half of the frog contour is corrupted even after a maxpooling layer. In comparison, the random errors only corrupted a small region and the frog contour is preserved. Hence, the burst errors are more likely to cause a large accuracy loss than random errors.

C. Characterizing error tolerance in FL

As mentioned in Sec. I, instead of feature maps, the clients in FL transfer the model weights. Hence, it is reasonable to expect that FL possesses different error tolerance characteristics than split ML. We investigate the impact of errors on a FL system with the VGG11 model. We assume a typical FL system with 20 participating devices, each training with $\frac{1}{20}$ CIFAR-100 dataset that follows i.i.d. distribution. The model weights are aggregated using the widely-adopted FedAvg algorithm [31] in a synchronized manner, *i.e.*, in each epoch, one client only uploads model weights one time. To avoid the impact of heterogeneous link conditions, we assume the FL system selects clients with similar link conditions and all clients share the same error rate. To investigate layer-wise error tolerance, we only apply errors to one layer's weights for each trial. Note that the activation and pooling layers are not considered in the experiment since they do not have weights. The model is trained over 50 epochs and then tested with the CIFAR-100 testing set. For each layer and error rate, we repeat the experiment 10 times. From the results, we make the following key observations:

Batch norm layers in FL are not error-tolerant. Fig. 6 shows the top-1 accuracy when applying random errors with 10% error rate to each layer. Although most layers maintain a stable accuracy for all 10 trials, layer 4, 8, 10, and 12 show a wide range of accuracy fluctuations up to 70%. Upon further inspection, we found that all layers with larger than 20% accuracy fluctuations are batch norm layers. Fig. 8 further shows the accuracy when the two batch norm layers experience different error rates. We see such fluctuations exist persistently.

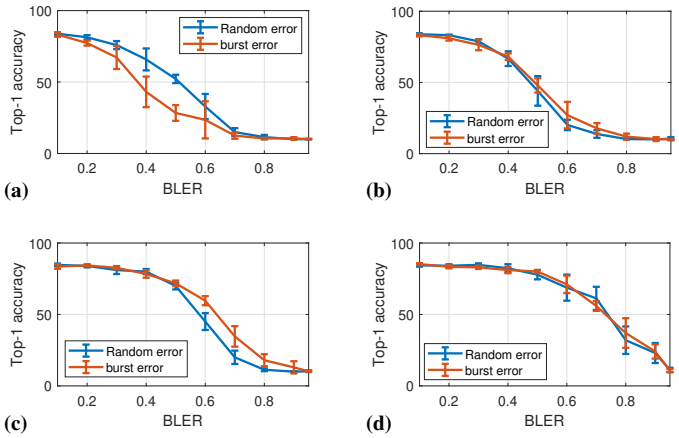

Fig. 7: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 8st, (b) 11th, (c) 18th, and (d) 22th layers (conv).

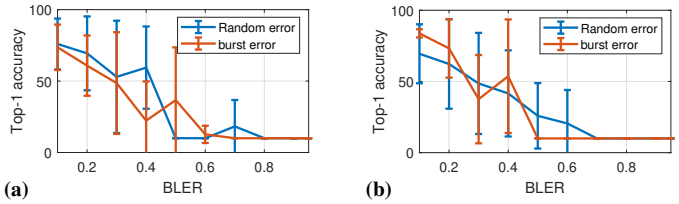

Fig. 8: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 16st, (b) 19th layers (batch norm).

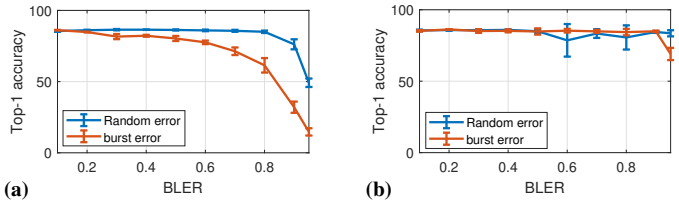

Fig. 9: Top-1 accuracy of VGG11 trained in FL under different block error rate applied to (a) 25th, (b) 26th layers (fully connected).

Unlike the convolution layers' weights which only affect one small region of a feature map, the weights in the batch norm layers are applied to a batch of feature maps. Consequently, the weight errors have a larger impact on accuracy. This result indicates that *ideally batch norm layers' weights should be transferred error-free*.

FL is prone to both burst errors and random errors. To investigate the impact of error patterns, we compare the accuracy from two different types of error patterns. As shown in Fig. 7 and 9, we see that for both convolution and fully connected layers, random error and burst error have similar impacts on model accuracy. This is because the training process in ML has a forward phase and a backward phase. The forward phase, essentially the same as the inference phase, computes the inference result and the corresponding loss from the first layer to the last layer, while the backward phase uses the loss to

compute the gradient of each layer propagating from the last layer to the first layer. Similar to split ML, the convolution and pooling layers dilute the errors in the forward phase and hence reduces its negative impact on the model accuracy. However, the same layers spread out errors to more parameters during the backward phase which amplifies the impact of errors on the model accuracy. As a result, there is no discernible advantage of random errors over burst errors. This means *FL's error tolerance cannot be improved by interleaved coding*.

V. NeuroMessenger Operations

A. Error tolerance Enhancing Coding

Interleaved Coding The foregoing experiments hint that split ML suffers more accuracy loss from burst errors than random errors under the same error rate. In practice, however, due to the packetization operation and wireless channel coherence, the majority of the errors are in the form of burst errors [32]. To alleviate the impact of such burst errors, we adopt an interleaved coding method to encode the feature maps. Interleaved coding is a family of codes aiming to convert the burst errors to random errors by interleaving the data sequence [33]. The basic idea is that the interleaving operation redistributes a long sequence of errors across many short separated bursts.

To showcase the interleaved coding, we apply it to the ML data in the burst error experiment in Sec. IVB and compare the top-1 accuracy. Without loss of generality, we use random interleaved coding [34]. The code generates a randomized permutation whose size equals the number of float point numbers in ML data and reorders the numbers in the data accordingly. To understand the result across different models, we add VGG11 [35], another popular image classification NN, to the experiment. We choose 4 different split points for each model, each at a quadrate point of the model. Fig. 10 and 11 show the the results.

We see that interleaved coding improves accuracy by up to 190% for all split points in ResNet18 and first 3 points in VGG11 (we will explain the last point later). Specifically, the last two points in ResNet18 show less than 1% accuracy loss when error rate is less than 50% while the baseline loses 45% accuracy at the same error rate. The result implies *the interleaving technique substantially improves the burst error tolerance for most layers in CNN models*.

Importance-based Coding Unlike the regional operations in the convolution and pooling layers, a fully connected layer operates on all parameters in feature maps and the output is a linear combination of all parameters whose weights are obtained during the training phase. Instead of the patterns and shapes in feature maps, the impact on accuracy from fully connected layers is determined by the absolute value of individual parameters. Recall that interleaved coding only redistributes the long sequence of errors without reducing the number of errors. As a result, the *interleaved coding does not improve the error tolerance for fully connected layer*. This explains the result in Fig. 11(d) where the interleaved coding shows similar accuracy as the baseline.

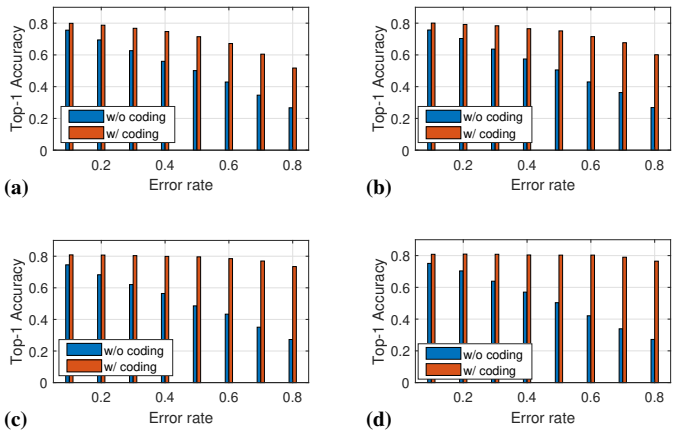

Fig. 10: Top-1 accuracy of ResNet18 under different block error rate with the split point after (a) first, (b) second, (c) third, and (d) forth residual module.

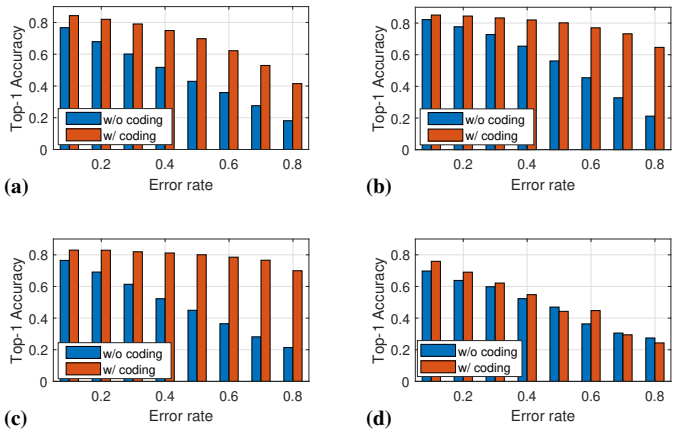

Fig. 11: Top-1 accuracy of VGG11 under different BLER with the split point after (a) 1st, (b) 5th, (c) 19th, and (d) 26th layer.

To enhance error tolerance for fully connected layers, we adopt importance-based coding, a family of codes that provide Unequal Error Protection (UEP) capability for data with different importance. The basic idea is that a corrupted high-valued parameter will have more impact on accuracy than a corrupted low-valued parameter (*e.g.*, near 0). Hence the high valued parameters are more important and should be protected against high error rates by UEP.

To showcase the importance-based coding, we repeat the previous burst error experiment for the DeepSpeech2 model [36]. DeepSpeech2 is a recurrent neural network (RNN) speech recognition model consisting of fully connected and activation layers. Similar to ResNet18, it is often used as a benchmark to evaluate distributed ML systems [37]. To show the maximum possible improvement, we assume an ideal importance-based coding scheme that ensures the error rate of a particular parameter is inversely proportional to its value while the total error rate is constant. In practice, the importance coding applies UEP techniques such as Hadamard matrix [38], network slicing [39], and repetition [40], to the high-valued parameters to

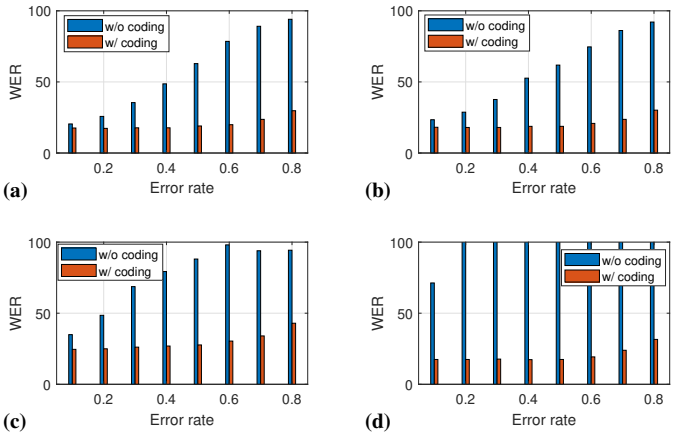

Fig. 12: Word error rate of DeepSpeech2 under different block error rate with the split point after (a) first, (b) second, (c) third, and (d) forth splitting point.

reduce their error rate under the same overall error rate. Fig. 12 shows the Word Error Rate (WER) performance for 4 splitting points at each quadrate point. A higher WER means lower speech recognition accuracy We see the ideal importance-based coding generally reduces WER by up to 60%. When block error rate (BLER) is less than 50%, the WER with ideal importance-based coding is maintained at $< 20\%$, less than 1% increase from the 0% block error rate case. The result shows that the ideal *importance-based coding effectively enhances the error tolerance for fully connected layers*.

B. Additional Operations

Layer-wise error-tolerance profiling As demonstrated in Sec. IV and IV-C, the accuracy performance of a distributed ML model depends on the link error rate, as well as on which layer the error occurs. To ensure a certain level of accuracy, we need to profile the mapping between the error rate and inference accuracy, and use the profile to estimate accuracy under a given link condition at runtime. We first reuse the experimental settings in Sec. IV and IV-C where the ML data experience random errors. For split ML models, since the splitting point may change at runtime, we profile the error rate to accuracy mapping for each layer offline. For FL models, we only profile a single error rate to accuracy mapping by applying the errors to the weights of all except the batch norm layers. We vary the error rate and record the corresponding top-1 testing accuracy. Suppose the error rates are $\{r_1, r_2, ..., r_L\}$, with corresponding accuracy $\{a_1, a_2, ..., a_L\}$. To approximate the error rate to accuracy mapping function, we empirically choose an exponential function G to fit the mapping, so that the L1 distance between the inferred accuracy and measured accuracy is minimized.

$$G = \min_{G} \sum_{i=1}^{L} |a_i - G(r_i)| \tag{2}$$

Since the error tolerance strongly depends on error rate (Sec. IV), we can thus accurately estimate the accuracy with G, for a given error rate (derived from the link SNR). The profiling is performed offline and only needs to be done once for a given ML model. The profiling latency is determined by the device's computation power as well as the size and depth of the ML model. We observe a profiling latency of $< 10s$ for split ML and $< 10min$ for FL with 54-layer ResNet18 model on a server with Nvidia RTX2080 TI GPU.

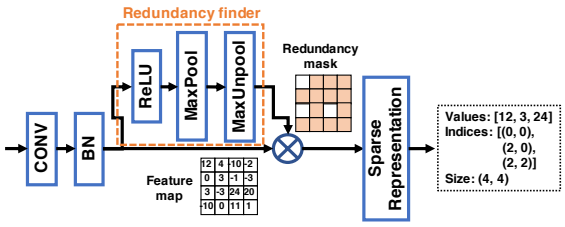

Fig. 13: An illustration of feature map pruning: (1) generate redundancy mask, (2) Multiply redundancy mask to the feature maps, (3) Convert pruned feature maps to sparse representation.

Feature map pruning The aforementioned error-tolerance techniques effectively reduce the need for frequent retransmissions, but do not reduce the data size which may induce a large initial transmission overhead. To further improve efficiency, we introduce a feature map pruning design for split ML. It is observed that most parameters (*i.e.* the float point numbers) in the feature maps are dropped by the first maxpooling and activation layers after the split point, *i.e.*, their values do not affect the inference result of the ML model [41]. Hence, these redundant parameters can be safely removed to reduce the ML data size. However, it is challenging to determine which parameters are redundant since it depends on the model input and cannot be predicted without running the layers first.

To efficiently find the redundant parameters, we introduce a *quantized NN based redundancy prediction* scheme. The general idea is that since the parameters are made redundant by the first activation and pooling layers after the split point, we can feed the feature maps to such layers to find which parameters are redundant. It is time-consuming to run full layers so instead, we use their quantized versions which have the same operations. As shown in Fig. 13, when a feature map is generated at the split point, it is first quantized and then fed to a redundancy finder branch, which consists of the first activation and pooling layers after the split point, and an unpooling layer [42] to map the pooling layer's output to their corresponding positions in the original feature maps. The output of the redundancy finder branch is a feature map that has the same size as the original feature map, but with redundant parameters converted to zero. We replace all non-zero parameters in this feature map output to 1 and multiply it element-wisely with the original feature maps. The feature map becomes sparse after such a pruning step and can be efficiently stored using standard sparse representation [43] with a data size of only a fraction of the original feature maps.

VI. Evaluation

A. Experimental setup

We evaluate NeuroMessenger on a Matlab-based 5G link-level simulation framework [44], which simulates an end-to-end

TABLE I: The top-1 inference accuracy and the end-to-end latency performance of NeuroMessenger split ML and baselines under a Matlab simulated noisy 3GPP NR uplink.

	Split point 1			Split point 2			Split point 3			Split point 4		
	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?
VGG11 - CIFAR100												
Ours	36.0ms (-91.9%)	79.7% (-8.1%)	None	26.7ms (-87.7%)	83.3% (-3.1%)	None	4.8ms (-91.2%)	82.0% (-11.1%)	None	<1ms (-93.0%)	65.8% (-23.5%)	None
Raw	433.8ms (-0.0%)	68.2% (-20.7%)	None	216.8ms (-0.0%)	85.2% (-0.9%)	None	54.3ms (-0.0%)	83.0% (-3.5%)	None	13.6ms (-0.0%)	65.4% (-24.0%)	None
Pruned	33.8ms (-92.2%)	60.14% (-30.1%)	None	24.5ms (-88.7%)	72.8% (-15.3%)	None	2.6ms (-95.2%)	61.4% (-28.6%)	None	< 1ms (-93.0%)	59.8% (-30.5%)	None
Pruned-HARQ	46.6ms (-89.3%)	86.0% (-0.0%)	Yes	38.0ms (-82.4%)	86.0% (-0.0%)	Yes	4.1ms (-92.4%)	86.0% (-0.0%)	Yes	< 1ms (-93.0%)	86.0% (-0.0%)	Yes
ResNet18 - CIFAR100												
Ours	16.0ms (-39.6%)	76.8% (-4.0%)	None	6.0ms (-55.9%)	78.4% (-2.0%)	None	2.5ms (-63.2%)	80.0% (-0.0%)	None	2.1ms (-36.4%)	80.0% (-0.0%)	None
Raw	26.5ms (-0.0%)	66.5% (-16.9%)	None	13.6ms (-0.0%)	78.6% (-1.8%)	None	6.8ms (-0.0%)	68.3% (-14.6%)	None	3.3ms (-0.0%)	68.3% (-14.6%)	None
Pruned	14.9ms (-43.8%)	76.8% (-4.0%)	None	3.8ms (-72.1%)	63.7% (-20.4%)	None	2.2ms (-67.6%)	62.0% (-22.5%)	None	1.7ms (-48.5%)	63.9% (-20.1%)	None
Pruned-HARQ	23.1ms (-12.8%)	80.0% (-0.0%)	Yes	6.0ms (-55.9%)	80.0% (-0.0%)	Yes	3.3ms (-51.5%)	80.0% (-0.0%)	Yes	2.6ms (-21.2%)	80.0% (-0.0%)	Yes
DeepSpeech2 - LibriSpeech (Accuracy represented by WER)												
Ours	3.5ms (-62.8%)	17.7% (+0.5%)	None	3.5ms (-62.8%)	18.0% (+2.2%)	None	3.5ms (-62.8%)	25.0% (+42.0%)	None	3.5ms (-62.8%)	17.7% (+0.5%)	None
Raw	9.4ms (-0.0%)	22.3% (+26.7%)	None	9.4ms (-0.0%)	27.6% (+56.8%)	None	9.4ms (-0.0%)	47.2% (+168.2%)	None	9.4ms (-0.0%)	80.1% (+355.1%)	None
Pruned	3.0ms (-62.8%)	35.4% (+101.1%)	None	3.0ms (-62.8%)	37.6% (+113.6%)	None	3.0ms (-62.8%)	68.7% (+290.3%)	None	3.0ms (-62.8%)	100.0% (+468.2%)	None
Pruned-HARQ	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes	4.6ms (-51.1%)	17.6% (-0.0%)	Yes

TABLE II: The top-1 inference accuracy and the end-to-end latency performance of NeuroMessenger FL and baselines under a Matlab simulated noisy 3GPP NR uplink.

	VGG11 - CIFAR100			ResNet18 - CIFAR100			DeepSpeech2 - LibriSpeech		
	t_{comm}	Acc.	retr. ?	t_{comm}	Acc.	retr. ?	t_{comm}	WER	retr. ?
Ours	8.80s (-35.6%)	84.0% (-2.3%)	None	0.77s (-35.8%)	82.6% (-4.0%)	None	20.12s (-38.8%)	18.7% (-6.3%)	None
HARQ	13.66s (-0.0%)	86.0% (-0.0%)	Yes	1.20s (-0.0%)	80.0% (-0.0%)	Yes	31.36s (-0.0%)	17.6% (-0.0%)	Yes

5G NR uplink with complete MCS implementations as well as 3GPP compliant channel models. By default, we use 16QAM modulation with 490/1024 LDPC code rate. Varying the link SNR leads to different levels of error rate. The OFDMA uses 20MHz bandwidth with 30KHz subcarrier spacing. We adopt a 2×2 MIMO setting with 2 PUSCH layers. The transport block size is set to 30216 bits. For the physical channel, we use 3GPP CDL-C clustered delay line channel [45] which represents a generic multi-path channel. Specifically, we set the SNR of the channel to -2dB by default to simulate a noisy link. Using a high order MCS, even with a relatively high SNR, may result in a similarly noisy link.

Models and dataset. We evaluate NeuroMessenger on two of the most widely deployed applications: image classification and speech recognition. For image classification, we choose ResNet18 and VGG11 to represent state-of-art CNN models. For testing purposes, we use CIFAR100, a standard image classification data set with 100 classes and 100 test images for each class. For speech recognition, we use the DeepSpeech2 model applied on the LibriSpeech dataset, a large corpus of reading English speech containing 5 hours of testing speech.

Baselines For split ML experiments, we compare the performance of NeuroMessenger with the following three baselines: *(i) Raw* does not perform any additional processing on the ML data. *(ii) Pruned* prunes the redundancy of the ML data but does not perform the two coding schemes that enhance error tolerance. *(iii) Pruned-HARQ* prunes the redundancy and leverages the built-in HARQ retransmission mechanism in 5G. The retransmission version (RV) in HARQ, *i.e.*, the maximal number of retransmission attempts, is set to 16. For FL experiments, since feature map pruning and error-tolerant enhancement schemes are not applicable, we use only one baseline: *HARQ*, *i.e.*, the default retransmission scheme in 5G instead of the retransmission controller in NeuroMessenger.

B. End-to-end performance

Table I and II show the inference accuracy and end-to-end latency performance for split ML and FL. In this experiment, we set the maximal tolerable accuracy loss of NeuroMessenger to 80% (-6% compared to the original model) for image classification models, and the maximal WER increase to 20% ($+3.4\%$ comparing to the original model) for the speech recognition model. Compared to the raw baseline, the pruned baseline reduces up to 95.2% transmission times accompanied by 9.3% to 113.1% higher loss in accuracy. This means *NeuroMessenger's redundancy pruning design effectively reduces the intermediate data size*. But it sacrifices the error

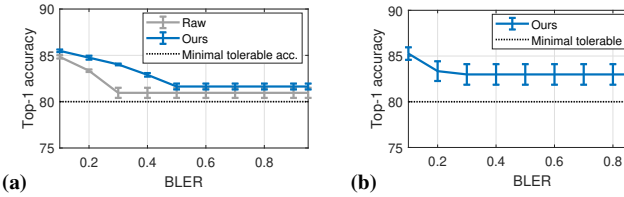

Fig. 14: Top-1 accuracy of (a) split ML (VGG11 split at 15-th layer) and (b) FL (VGG11), as the block error rate of the link increases.

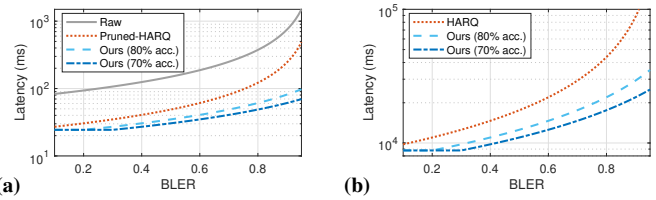

Fig. 15: End-to-end running latency of (a) split ML (ResNet18 split at 15-th layer) and (b) FL (ResNet18), as the block error rate of the link increases

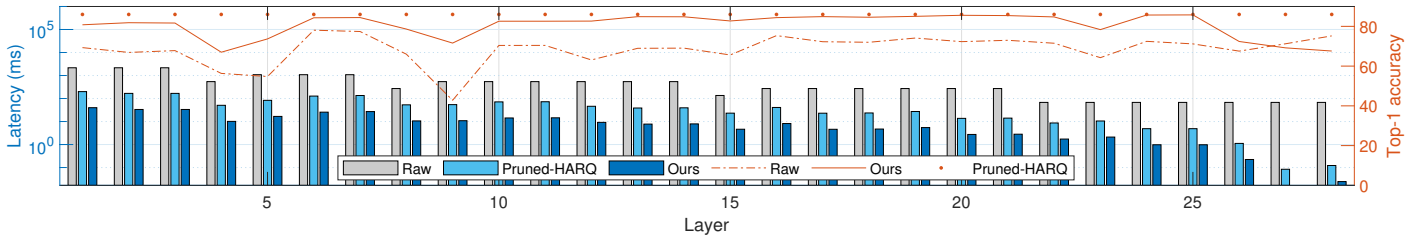

Fig. 16: End-to-end running latency and top-1 accuracy at BLER=0.2 when splitting at each layer in VGG11.

tolerance capability, implying the need for error enhancing coding. The pruned-HARQ baseline achieves the best accuracy due to frequent retransmissions which also causes 20% more communication latency than the pruned baseline. *Our NeuroMessenger design strikes the best balance between the accuracy and latency*: its transmission time only increases 1%-5% on top of the pruned baseline due to the computation time of coding while the accuracy is maintained above the maximal tolerable accuracy most of time, which is 0.9%-30.5% higher than the raw and pruned baselines. In summary, the experiment shows the performance advantages of NeuroMessenger over the baselines in split ML and the necessity of the coding mechanisms for enhancing error-tolerance.

C. Impact of Link Conditions

To investigate the performance of NeuroMessenger under different link conditions, we vary the SNR of the 5G link so that the block error rate increases from 0.1 to 0.95. Fig. 14 and Fig. 15 show the top-1 accuracy and latency results of ResNet18 in split ML and FL, respectively. We set the minimum tolerable accuracy to 80% for both split ML and FL. In Fig. 14, we see accuracy slightly decreases to 82% as the BLER increases. It remains at the same level as the retransmission controller determines that error tolerance can no longer satisfy the accuracy requirement and enables retransmission. Note that even for the raw baseline in Fig. 14, the NeuroMessenger retransmission controller can still guarantee a minimal latency of 80% by enabling retransmission earlier.

In Fig. 15(a), we see the latency of raw and pruned-HARQ baselines increases exponentially with BLER as the number of retransmission increases. The latency of NeuroMessenger stays constant until BLER$=0.5$ due to the absence of retransmissions. Note that if we lower the minimal tolerable accuracy to 70%, the latency stays constant until BLER$=0.9$. The FL's latency in Fig. 15(b) shows a similar trend but with a lower constant latency.

In summary, this experiment shows that *the retransmission controller can effectively guarantee the accuracy under a wide range of block error rate, and NeuroMessenger keeps the communication latency constant when the block error rate is within the error tolerance of the ML data*, which is determined by the ML model, layer, and minimal tolerable accuracy.

D. Impact of Split Point

Recall that in split ML, the error tolerance is different across layers (Sec. IV). Hence the effectiveness of NeuroMessenger may vary with different choices of split point. To investigate this effect, we reuse the experimental setup in Sec. VI and split a VGG11 model at each layer. Fig. 16 shows the latency and corresponding accuracy performance. We see that NeuroMessenger achieves 40%-99% latency reduction comparing to the raw baseline and 20% comparing to the pruned-HARQ baseline for all layers. Although pruned-HARQ achieves the highest accuracy, NeuroMessenger achieves similar accuracy for most layers and significantly higher accuracy than the raw baseline. In summary, this experiment and the experiment in Sec. V-A jointly show that the advantage of NeuroMessenger applies to a wide range of split points on typical NN models.

VII. Conclusion

In this paper, we have explored the error tolerant capability in distributed ML data transfer and its implications on communication efficiency. We characterize the error tolerance of various popular distributed ML systems and develop a novel system, NeuroMessenger that enhances and utilizes such error tolerance. We believe our work envisions a new direction towards efficient distributed ML over wireless edge networks.

Acknowledgement

The work reported in this paper is supported in part by a Sony Research Award, and by the NSF under Grants CNS-1901048, CNS-1925767, CNS-1952942, and CNS-2128588.

REFERENCES

[1] S. Kekki, W. Featherstone, Y. Fang, P. Kuure, A. Li, A. Ranjan, D. Purkayastha, F. Jiangping, D. Frydman, G. Verin *et al.*, "Mec in 5g networks," *ETSI white paper*, vol. 28, pp. 1–28, 2018.

[2] E. Peltonen, M. Bennis, M. Capobianco, M. Debbah, A. Ding, F. Gil-Castiñeira, M. Jurmu, T. Karvonen, M. Kelanti, A. Kliks *et al.*, "6g white paper on edge intelligence," *arXiv preprint arXiv:2004.14850*, 2020.

[3] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] 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.

[6] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive dnn surgery for inference acceleration on the edge," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1423–1431.

[7] 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.

[8] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, 2018, pp. 31–36.

[9] J. Chen and X. Ran, "Deep learning with edge computing: A review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.

[10] I. Stoica, D. Song, R. A. Popa, D. Patterson, M. W. Mahoney, R. Katz, A. D. Joseph, M. Jordan, J. M. Hellerstein, J. E. Gonzalez *et al.*, "A berkeley view of systems challenges for ai," *arXiv preprint arXiv:1712.05855*, 2017.

[11] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

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

[13] F. Boccardi, R. W. Heath, A. Lozano, T. L. Marzetta, and P. Popovski, "Five disruptive technology directions for 5g," *IEEE Communications Magazine*, vol. 52, no. 2, pp. 74–80, 2014.

[14] 3GPP, "5g system (5gs); study on traffic characteristics and performance requirements for ai/ml model transfer," TR 22.874 V0.0.0, 2020.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] S. U. Stich, J.-B. Cordonnier, and M. Jaggi, "Sparsified sgd with memory," in *Advances in Neural Information Processing Systems*, 2018, pp. 4447–4458.

[17] 3GPP, "Nr; study on integrated access and backhaul," TR 38.874 V16.0.0, 2019.

[18] M. G. Arivazhagan, V. Aggarwal, A. K. Singh, and S. Choudhary, "Federated learning with personalization layers," *CoRR*, vol. abs/1912.00818, 2019.

[19] H. Wang, M. Yurochkin, Y. Sun, D. S. Papailiopoulos, and Y. Khazaeni, "Federated learning with matched averaging," *CoRR*, vol. abs/2002.06440, 2020.

[20] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–7.

[21] N. Yoshida, T. Nishio, M. Morikura, K. Yamamoto, and R. Yonetani, "Hybrid-fl for wireless networks: Cooperative learning mechanism using non-iid data," in *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, 2020, pp. 1–7.

[22] T. T. Anh, N. C. Luong, D. Niyato, D. I. Kim, and L.-C. Wang, "Efficient training management for mobile crowd-machine learning: A deep reinforcement learning approach," *IEEE Wireless Communications Letters*, vol. 8, no. 5, pp. 1345–1348, 2019.

[23] J. Kang, Z. Xiong, D. Niyato, S. Xie, and J. Zhang, "Incentive mechanism for reliable federated learning: A joint optimization approach to combining reputation and contract theory," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 700–10 714, 2019.

[24] J. Kang, Z. Xiong, D. Niyato, H. Yu, Y.-C. Liang, and D. I. Kim, "Incentive design for efficient federated learning in mobile networks: A contract theory approach," in *2019 IEEE VTS Asia Pacific Wireless Communications Symposium (APWCS)*, 2019, pp. 1–5.

[25] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 476–488.

[26] P. Hu, J. Im, Z. Asgar, and S. Katti, "Starfish: resilient image compression for aiot cameras," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, 2020, pp. 395–408.

[27] M. Hamza, A. Lipovac, and V. Lipovac, "Residual block error rate prediction for ir harq protocol," *Tehnički vjesnik*, vol. 27, no. 4, pp. 1071–1076, 2020.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[29] 3GPP, "Nr; physical layer; general description," TS 38.201 V16.0.0, 2020.

[30] ——, "Nr; physical layer procedures for data," TR 38.214 V16.4.0, 2021.

[31] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-Efficient Learning of Deep Networks from Decentralized Data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. PMLR, 20–22 Apr 2017, pp. 1273–1282.

[32] C.-Y. Hsu, A. Ortega, and M. Khansari, "Rate control for robust video transmission over burst-error wireless channels," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 756–773, 1999.

[33] S. Lin and D. J. Costello, *Error control coding*. Prentice hall, 2001, vol. 2, no. 4.

[34] S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah, "Randomized gossip algorithms," *IEEE transactions on information theory*, vol. 52, no. 6, pp. 2508–2530, 2006.

[35] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[36] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. H. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015.

[37] S. Yao, J. Li, D. Liu, T. Wang, S. Liu, H. Shao, and T. Abdelzaher, "Deep compressive offloading: Speeding up neural network inference by trading edge computation for network latency," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, ser. SenSys '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 476–488. [Online]. Available: https://doi.org/10.1145/3384419.3430898

[38] S. Jakubczak and D. Katabi, "A cross-layer design for scalable mobile video," in *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 289–300.

[39] ETSI, "5g;management and orchestration; architecture framework," ETSI TS 128 5ss V16.4.0, 2020.

[40] M. Kanj, V. Savaux, and M. Le Guen, "A tutorial on nb-iot physical layer design," *IEEE Communications Surveys & Tutorials*, 2020.

[41] S. Cao, L. Ma, W. Xiao, C. Zhang, Y. Liu, L. Zhang, L. Nie, and Z. Yang, "Seernet: Predicting convolutional neural network feature-map sparsity through low-bit quantization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 216–11 225.

[42] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[43] PyTorch, "torch.sparse — pytorch 1.9.0 documentation," 2020. [Online]. Available: https://pytorch.org/docs/stable/sparse.html

[44] MathWorks, "Nr pusch throughput," 2020. [Online]. Available: https://www.mathworks.com/help/5g/ug/nr-pusch-throughput.html

[45] 3GPP, "Study on channel model for frequency spectrum above 6 ghz," TR 38.900 V15.0.0, 2018.