DISCO: Distributed Inference with Sparse Communications

Minghai Qin
Western Digital
minghai.qin@wdc.com

Chao Sun
Western Digital
chao.sun@wdc.com

Jaco Hofmann
Western Digital
jaco.hofmann@wdc.com

Dejan Vucinic
Western Digital
dejan.vucinic@wdc.com

Abstract*—Deep neural networks (DNNs) have great potential to solve many real-world problems, but they usually require an extensive amount of computation and memory. It is of great difficulty to deploy a large DNN model to a single resource-limited device with small memory capacity. Distributed computing is a common approach to reduce single-node memory consumption and to accelerate the inference of DNN models. In this paper, we explore the "within-layer model parallelism", which distributes the inference of *each layer* into multiple nodes. In this way, the memory requirement can be distributed to many nodes, making it possible to use several edge devices to infer a large DNN model. Due to the dependency within each layer, data communications between nodes during this parallel inference can be a bottleneck when the communication bandwidth is limited. We propose a framework to train DNN models for Distributed Inference with Sparse Communications (DISCO). We convert the problem of selecting which subset of data to transmit between nodes into a model optimization problem, and derive models with both computation and communication reduction when each layer is inferred on multiple nodes. We show the benefit of the DISCO framework on a variety of CV tasks such as image classification, object detection, semantic segmentation, and image super resolution. The corresponding models include important DNN building blocks such as convolutions and transformers. For example, each layer of a ResNet-50 model can be distributively inferred across two nodes with 5x less data communications, almost half overall computations and less than half memory requirement for a single node, and achieve comparable accuracy to the original ResNet-50 model.

I. Introduction

Deep neural networks (DNNs) have made great progress in solving real-world problems [1]. A majority of studies assume that DNN models are inferred on a single device, such as a GPU. There are a few reasons to study the distributed inference of DNN models on multiple devices. Firstly, good DNN models usually require large memory for inference. This issue becomes more critical as the size of modern DNN models becomes larger and the popularity of resource-limited devices for them to be deployed grows rapidly. For example, ESP32 [2] series of SoCs have only hundreds of kilobytes of RAM/ROM, while a ResNet50 model requires hundreds of megabytes of RAM to store its weights and features, making it extremely challenging to infer a large DNN model on the edge device. This difficulty can be mitigated by distributing the memory consumption of DNN models to multiple nodes. Second, the inference latency can also be reduced by distributing the computation of inference to multiple nodes as well. Thirdly, some applications can benefit from cooperative inference of several branches DNN models. For example, in the scenario of multi-camera surveillance, if the DNN model in each camera can not only infer features from its own view, but collaboratively receive/send data from/to other cameras, the detection of an object can be more accurate. The overall system can be viewed as a distributed inference of a larger DNN model with communicated features between nodes.

Parallelism by distributing the computation across many nodes is one of the most popular methods to accelerate the DNN and it can also reduce the memory requirement for each node [6], [7]. In the *data parallelism*, each node is responsible for processing a subset of input samples in one mini-batch. In the *pipeline parallelism* [8]–[11], the DNN model is partitioned into sequential parts based on the execution order, and each node is responsible for processing one part. The DNN is therefore processed in the pipeline manner. Although they improve throughput, neither data nor pipeline parallelism can reduce the latency during the inference of one input. This is because the input data has to be sequentially processed through all DNN layers, and within each layer the computation is not distributed.

In order to distribute the computation within a layer, the *model parallelism* is proposed where each node is responsible for computing a subset of the output of a layer [7]. Note that each feature in the output of a layer is usually dependent on **all** input data of the layer, such that all input data needs to be distributed to all nodes (Figure 1(a)). The data communication latency between nodes will be the bottleneck of the distributed system with low communication bandwidth. For example, based on the configuration of a head-mounted system in [4], the communication latency can be one or two orders of magnitudes higher than the computation latency (See Figure 2). This phenomenon also happens for high-end (e.g., A100 GPUs and NV-Links) and low-end (e.g., ARM cores and wireless links) platforms. Many prior works [12]–[16] tried to use better scheduling methods to improve the overall system performance. However, the potential of jointly designing the DNN architectures based on the system configurations is not explored. Another trend to distribute the computation is by splitting some stages of the DNN into independent branches [3]–[5] (Figure 1(b)(c)(d)). In these works, the separate branches can be processed independently

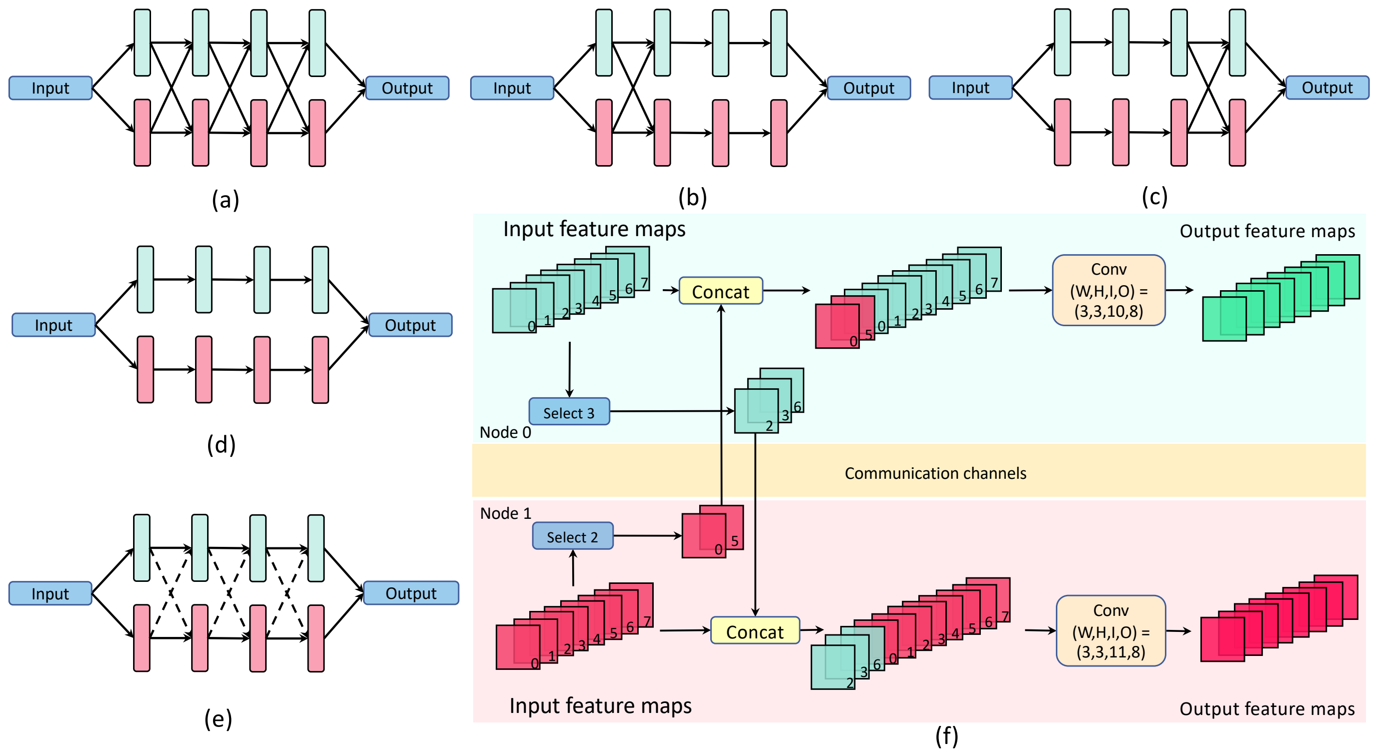

Fig. 1: Comparison of different distributed DNN inference methods. (a) Conventional dense communications. (b) Dense-then-split ([3]). (c) Split-then-aggregate ([4]). (d) Independent branches ([5]). (e) Distributed inference with sparse communications (DISCO, ours). Dashed arrows mean sparse data transmission. (f) Details of one layer in an example 2-node DISCO. Each node has 8 features from previous layers. Node-0 and Node-1 select 3 and 2 data features (out of 8) to transfer to the other node, respectively. The transferred features are combined with its own features to form the input to the convolutional kernels. The output of the convolution (after point-wise non-linear activation functions) will again be served as the input to the next layer with sparse communications.

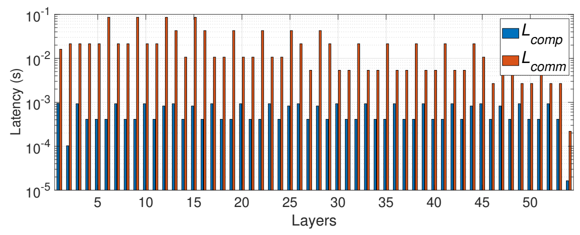

Fig. 2: Computation latency (L_{comp}) and communication latency (L_{comm}) of 54 layers (including 4 skip links) in a ResNet-50 with dense communications between $N = 2$ nodes. The system configuration is in [4], with $B = 37.5$ MB/s, $C = 125$ GOP/s.

on different nodes and there is no data communication between them. This eliminates the data communication latency for the corresponding stages. However, completely separated branches hurt the accuracy.

In this paper, we jointly design the DNN architecture and system configurations for distributed inference. In our proposal, only a subset of input features are transmitted between nodes (Figure 1(e)). The problem of selecting an optimal subset of them to communicate between nodes is NP-hard. We first demonstrate that this problem is equivalent to the problem of DNN weight pruning with certain patterns. To be more specific, weights in a convolutional or fully-connected layer can be represented as a 2-dimensional matrix M. And if Node-i does not need a specific input feature from Node-j to compute Node-i's output features, it is equivalent to setting a sub-row of appropriate size and location in M to be all-zero. Details are presented in Section III-B. In our framework, we first train a complete model with dense communication. Then we identify the non-zeros weights corresponding to the sparse features to communicate. We gradually sparsify the communication by fine-tuning the remaining weights. This framework, called *distributed inference with sparse communications (DISCO)*, can search for models with a better trade-off between data communication, computation, and accuracy. In addition, observing that the data communication and computation can be pipelined to hide their latency, the proposed method can further reduce the end-to-end system latency at almost no cost to accuracy. By experimenting on multiple machine learning tasks, including image recognition, object detection, semantic segmentation, and super-resolution, we conclude that 1) Compared to DNNs with completely independent branches, the accuracy can be significantly improved by a very tiny proportion of the data communications; 2) Compared to DNNs with completely dependent input-output layers (dense data communications), our proposed method can significantly reduce data communications with negligible accuracy loss.

The contribution of this paper is summarized as follows.

1). We propose a framework, called *DISCO*, to design distributed DNN model architectures with sparse communications.

2). We proposed structured pruning patterns of DNN weight pruning problems which the problem of selecting the subset of data to communicate (which is NP hard) can be converted to.

3). The proposed DISCO can reduce the data communication by an average 75% without accuracy degradation based on 5 machine learning tasks and models. As far as we know, this covers the widest range of applications among similar works.

II. Related Works

A. Distributed inference

Some prior works assume a device-cloud scenario where the layers of neural network are distributed among them [4], [17]–[20]. Many works focus on selecting the splitting point for separate device and cloud execution [8]–[11]. In these methods, the communication happens at the splitting point and no other communication latency is considered within each side. Our work, on the contrary, focuses on the fundamentals of distributed inference where no difference in the computational capabilities of processors is assumed and we consider the communication latency of **all** layers. Therefore, our work can also be used to improve their latency on either the device or the cloud side.

[3], [5], [21], [22] partition the neural network into several branches and each branch is distributed to a node for independent execution. Our work is different in that our network design is not completely separate such that there is communication between nodes. We can show that a tiny fraction of communication (e.g., 1%) can improve accuracy significantly with our framework.

Unlike works with changes to DNNs, there are some other works focusing on the system scheduling of inference without changing the models. [12] uses compression and dynamic scheduling to improve the throughput. [13] forms the splitting problem into a graph routing problem. [14] presents a framework to dispatch the inference data to compute nodes. [15], [16] minimizes the footprint in parallelism and enables dynamic workload distribution. [23] considers heterogeneous and non-linear device characteristics and proposes splitting and distributing methods. Compared to system design methods where the focus is to find a good schedule for unchanged models, our work is different, and can be superimposed on theirs as well, in that we modify the network architecture to directly reduce data communications to achieve a better accuracy-latency trade-off.

B. Distributed training

Model and pipeline parallelism can accelerate the distributed training. Graph partitioning [24]–[30] splits the training data into micro-batches and they are pipelined into the training devices. [31]–[33] split the tensors and place them onto different devices to distribute the computing. Data parallelism encounters the problem of high communication bandwidth for transmitting gradients. [34]–[38] propose methods to sparsify and quantize the gradients to reduce communication, respectively.

While the distributed training research focuses on improving the training throughput, our work, on the other hand, focuses on improving the **distributed inference latency** of each DNN layer by a slight architectural change.

III. Methodology

A. Problem statement

The basic operation of the DNN model inference is the multiplication of input features and weights. The input features are feature maps for convolutional neural networks (CNNs) and are neuron vectors/matrices for multi-layer perceptions (MLPs) and Transformers. Assume there are I input features. In order to distribute the computation of each DNN layer into N nodes, they can be equally distributed with $\frac{I}{N}$ input features per node. If an output feature in Node-i is dependent on input features in Node-j, the corresponding dependent input features are required to be transmitted from Node-j to Node-i. Figure 1(f) demonstrates an example of distributing a convolutional layer with 3×3 kernels into $N = 2$ nodes. The total number of input features is $I = 16$ and each node carries 8 of them. For conventional DNNs with dense communications shown in Figure 1(a), the convolution tensor on each node has the shape $(W, H, I, O) = (3, 3, 16, 8)$, where W and H are the kernel size, I and O are 16 input features and 8 output features, respectively. The total number of output feature maps is then $8 + 8 = 16$. Though the computation of the convolution operation is distributed across 2 nodes, the communication latency may be a bottleneck of the system since all 8 input features need to be transmitted across the communication channel in both ways.

In DISCO (Figure 1(f)), in order to reduce the communication latency, for example, Node-0 can select 3 input features, which are the 2nd, 3rd, and 6th feature maps, and sends them to Node-1, resulting in $\frac{5}{8}$ one-way communication reduction. Similarly, in the other direction, the selection of the 0th and 5th feature maps from Node-1 to Node-0 can reduce the data transmission by $\frac{6}{8}$. The problem of designing the DNN model for distributed inference can be formulated as the following few questions:

1) If the total number of features to be communicated is known, which subset of input features should be selected to transmit to other nodes?

2) If the subset of transmitted input features is determined, how should the weights of the DNN be trained?

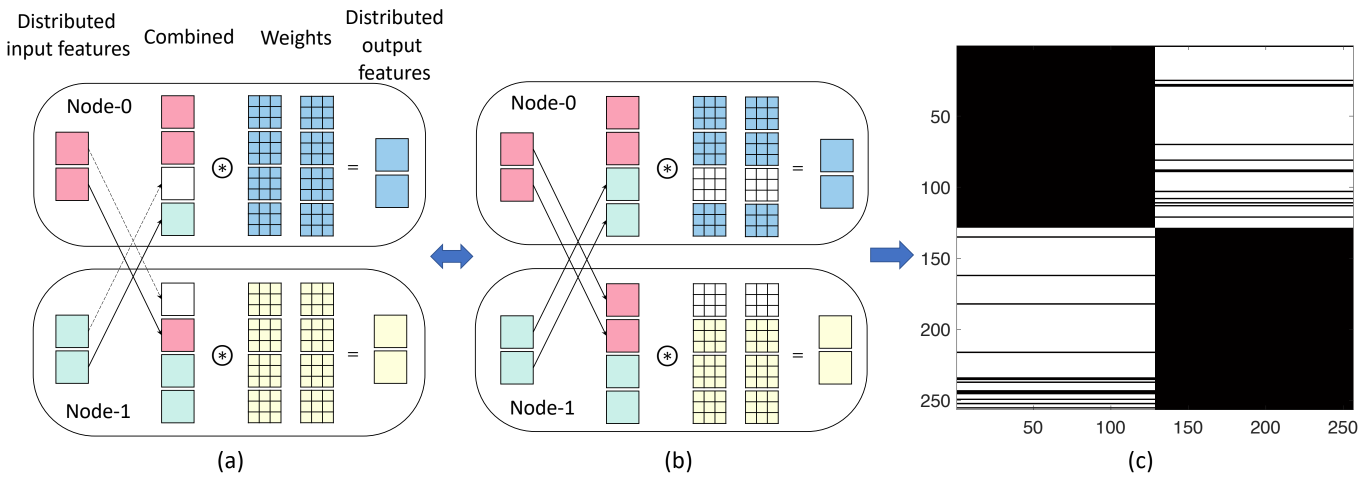

Fig. 3: Equivalence of feature selection and weight selection. (a) One of two features is transmitted from Node-0 to Node-1 and from Node-1 to Node-0, respectively. The non-transmitted feature can be viewed as all-zero (a blank feature). (b) The equivalent representation where all features are transmitted but the corresponding weights are all-zero (blank 3-by-3 kernels). (c) A real example of the weight matrix in a ResNet-50 model found by our DISCO. The number of both input and output feature maps are 256. Each black subrow in the anti-diagonal represents one transmitted input feature between the two nodes.

B. Selection of the subset of input features to communicate

In this section, we use a convolutional layer to demonstrate the subset selection problem and it can be generalized to MLP and Transformers as well.

Let $I_{\text{each}} = \frac{I}{N}$ be the number of local input features on each node, and assume the number of input features communicated from Node-i to Node-j is I_{comm}. The communication sparsity, denoted by $S_{\text{comm}} = 1 - \frac{I_{\text{comm}}}{I_{\text{each}}}$. Note that reduced feature communication also leads to computation reduction, denoted by $S_{\text{comp}} = \frac{N-1}{N} S_{\text{comm}}$ (See Appendix for the proof). There are $\binom{I_{\text{each}}}{I_{\text{comm}}}$ possible subsets of input features to select. This problem is NP-hard for moderate fraction $\frac{I_{\text{comm}}}{I_{\text{each}}}$. For example, if $I_{\text{each}} = 128$ and $I_{\text{comm}} = 32$ (transmitting a quarter of features, i.e., $S_{\text{comm}} = 75\%$), then the number of subset selections is larger than 10^{30}, making it impossible to enumerate.

Let the subset of input features that are *not* transmitted to the other node be denoted by $\mathcal{F}$, it is mathematically equivalent to transmitting them as all-zero features $\mathcal{F}_{\text{zero}}$ and the corresponding weights $W_{\mathcal{F}}$ convolving these all-zero features can be arbitrary (Figure 3(a)). This is because zero times any value equals zero. Then, it is mathematically equivalent to transmit all input features including $\mathcal{F}$ and set $W_{\mathcal{F}} = \mathbf{0}$, as in Figure 3(b). Therefore, the problem of selecting a subset of features, which is dynamic and related to the dataset distribution, can be converted into a weight pruning problem, where the pruning pattern is a sequence of convolution kernels (e.g., a sequence of 3×3 kernels) that corresponds to input features in the other nodes. Therefore, if the 4-D convolution weight tensor is projected to a 2-D mask-matrix by condensing each 2-D convolution kernel into one value representing whether this kernel is all-zero, then the pruned pattern is a half row of the matrix (for 2 nodes), or in general, a $\frac{1}{N}$-th row of the matrix, because each sub-row represents the weights that convolve one input feature in the other nodes. Note that for Node-i, the convolution kernels corresponding to its input features are always kept. Therefore, the diagonal blocks of the mask-matrix are always kept. Figure 3(c) shows the 2-node mask-matrix for one of the layers in ResNet-50 that has 256 input and output feature maps, respectively. The communication sparsity is 90%. Each point in the mask represents a 3×3 convolutional kernel. Two dark blocks on the diagonal represent the dense weights and communications within each node. In the anti-diagonal, white regions mean the corresponding weights are pruned and the dark black sub-rows correspond to those input features that will be communicated between the two nodes.

C. Training DNNs with sparse communications

After converting the input feature selection problem to a weight pruning problem, we can find an approximation of the problem efficiently by estimating the significance of the weights. First, a full DNN model with dense communications is trained. Second, for each layer, create a mask matrix $M \in \{0,1\}^{I \times O}$ for each convolutional kernel of shape $W \times H$ (e.g., 3×3 or 1×1), where I and O are the number of input and output features, respectively. Then, for the non-diagonal sub-row of size 1-by-$\frac{O}{N}$ that corresponds to $\frac{OWH}{N}$ weights in the full model, calculate the L_1 norm of them and prune the sub-rows with least L_1 norms to all-zero values. If we partition mask M into $\frac{I}{N} \times \frac{O}{N}$ sub-matrices, then pruning one non-diagonal sub-row in the (i, j)-th sub-matrix corresponds to preventing one input feature from being transmitted from Node-i to Node-j. The total number of pruned sub-rows depends on the desired sparsity in the non-diagonal matrix, which is the communication sparsity of this layer. The remaining sub-rows are the input features with greater significance and should be communicated between nodes for optimum latency-accuracy trade-offs. We ablate over the random sub-row pruning to show the benefit of DISCO.

In our experiment, we use the simple yet effective iterative magnitude prune-and-finetune procedure to obtain trained DNNs with sparse communications. First, we train a full model with dense communications. Then we prune the least significant sub-rows in the weight matrix by a fraction of p, and then finetune the remaining weights to recover some accuracy. This completes one iteration. We perform this fraction-p prune and finetune process for a few iterations to generate a sequence of models with increasing weight sparsity (and thus communication and computation sparsity).

IV. Experimental Results

In this section, we demonstrate the benefits of DISCO through several machine learning tasks, including image classification (the ResNet, EfficientNet and DeiT models), object detection (the SSD models), semantic segmentation (the DeepLabV3+ models) and super resolution (the ESRGAN models). All models are trained on NVIDIA A100 GPUs by PyTorch implementation.

In Tables I, II, III, IV, V, we investigate the communication and computation sparsity (denoted by S_{comm} and S_{comp}) and the accuracy trade-offs of various DNN models, datasets, and evaluation metrics when they are distributed across **two nodes**. Comparisons to a single node or larger number of nodes are presented in Section V. Details on the training recipe are presented in the Appendix. We vary S_{comm} (and thus S_{comp}) in the DISCO framework to obtain a sequence of DNN models.

We compare our framework (DISCO) with three different existing or baseline approaches.

1) Random sparse communications: The transmitted input features are randomly selected instead of being selected based on DISCO (Section III-B). In particular, when $S_{\text{comm}} = 1$, there is no communication between nodes and DNNs are partitioned into "independent branches" [5].

2) Dense-then-split: The features in the first few layers are densely communicated, and the remaining layers are all separate without feature communication (Figure 1(b)). Different splitting points result in different accuracy-latency trade-off. This architecture is proposed in [3].

3) Split-then-aggregate: The first few layers are all separate without communicated features, and the remaining layers have dense feature communications (Figure 1(c)). Different aggregation points result in different accuracy-latency trade-offs. This architecture is proposed in [4].

For DISCO and 1), we use uniform S_{comm} and S_{comp} across all layers. For 2) and 3), we report the average S_{comm} and S_{comp}.

The observation that our results outperform the random sparse communications shows the effectiveness of DISCO in selecting features and training the DNN (Section III-B, III-C). On the other hand, our results outperform "Dense-then-split" and "Split-then-aggregate", which shows that it is better to have sparse communications among all layers than to have polarized communication sparsity for different layers.

A. ResNet50 models on image classification

The ResNet architecture is widely used in computer vision tasks. We use ResNet-50 models on ImageNet [44] to demonstrate the benefit of DISCO.

A few conclusions can be drawn from Table I: **(1)** Compared to the dense model, DISCO methods has no accuracy loss when $S_{\text{comm}} = 80\%$, i.e., 5x data communication reduction. **(2)** The accuracy improvement over [3], [4] is significant at all sparsities. For example, the maximum S_{comm} of [3], [4] in the table is 91% with accuracy being 73.66%, while DISCO's accuracy is 76.5% with $S_{\text{comm}} = 90\%$. There is 2.8% accuracy gain. **(3)** Compared to the model with independent branches [5] ($S_{\text{comm}} = 100\%$), by adding only 1% feature communications ($S_{\text{comm}} = 99\%$), the accuracy can be improved by a large margin from 73.36% (without DISCO) to 75.94% (with DISCO). This demonstrates that a small portion of communicated features can significantly increase the model accuracy.

B. DeiT models on image classification

There is a growing interest in Vision Transformers [40], [45] and we also apply the DISCO to DeiT-small models. We follow the same training recipe as in [46]. Some conclusions can be drawn from Table II: **(1)** DISCO has almost no accuracy loss with 60% to 80% communication reduction. **(2)** DISCO has over 1% accuracy advantage over all existing or baseline methods. **(3)** By adding 1% of data communications between the two nodes, DISCO methods can improve the accuracy from 74.58% to 78.70%.

C. SSD models on object detection

The single-shot detection (SSD) model [47] is a fast one-phase object detection model. We use the COCO2017 [48] dataset and follow the training recipe in [41].

Some conclusions can be drawn from Table III: **(1)** DISCO has no accuracy loss with 90% communication reduction. **(2)** DISCO has clear accuracy advantages over prior art or baseline methods (+2.9% or more mAP at the same S_{comm}). **(3)** By introducing 1% data communication, the accuracy can be significantly improved from 19.6% to 24.6%.

D. DeepLabV3+ models on semantic segmentation

We use the standard DeepLabV3+ models with ResNet-101 as the backbone [42] for semantic segmentation task. The dataset is PASCAL VOC2012 [49]. We follow the training recipe in [42].

Some conclusions can be drawn from Table IV: **(1)** DISCO has no accuracy loss with 95% communication reduction. **(2)** DISCO has over 4-12% mIoU advantage over all existing or baseline methods. **(3)** By introducing 1% data communication, the accuracy can be improved from 64.5% to 76.7%.

TABLE I: Results of ResNet-50 models on ImageNet.

Method	S_{comm}	S_{comp}	Top-1 accu. (%)	Comments
Dense [39]	0%	0%	76.80	
Random Sparse Communication (baseline)	80%	40%	75.26	Uniform S_{comm} and S_{comp}
	90%	45%	74.90	
	95%	47.5%	74.54	
	99%	49.5%	73.85	
	100%	50%	73.36	
Dense-then-split [3]	8%	10%	75.14	Split @ Loc. 1
	32%	28%	74.55	Split @ Loc. 2
	68%	40%	73.3	Split @ Loc. 3
Split-then-aggregate [4]	30%	8%	75.10	Aggr. @ Loc. 1
	67%	21%	74.88	Aggr. @ Loc. 2
	91%	39%	73.66	Aggr. @ Loc. 3
DISCO (Ours)	80%	40%	**76.84**	Uniform S_{comm}
	90%	45%	**76.50**	
	95%	47.5%	**76.26**	
	99%	49.5%	**75.94**	

TABLE II: Results of DeiT-S models on ImageNet.

Method	S_{comm}	S_{comp}	Top-1 accu. (%)	Comments
Dense [40]	0%	0%	79.94	
Random Sparse Communication (baseline)	60%	30%	78.51	Uniform S_{comm}
	80%	40%	78.45	
	90%	45%	78.24	
	95%	47.5%	75.65	
	100%	50%	74.57	
Dense-then-split [3]	24%	15%	78.39	Split @ Loc. 1
	47%	30%	76.51	Split @ Loc. 2
Split-then-aggregate [4]	24%	15%	79.26	Aggr. @ Loc. 1
	47%	30%	78.51	Aggr. @ Loc. 2
DISCO (Ours)	60%	30%	**80.08**	Uniform S_{comm}
	80%	40%	**79.50**	
	90%	45%	**79.21**	
	95%	47.5%	**79.11**	
	99%	49.5%	**78.70**	

E. *ESRGAN models on super resolution*

We use the ESRGAN [43] with 23 residual-in-residual dense blocks (RRDB) [43] architecture to demonstrate the DISCO on image super resolution (up-scaling the height and width each by 4 times). The dataset is DIV2K [50].

Some conclusions can be drawn from Table V: **(1)** DISCO has no PSNR loss with 60%-80% communication reduction. **(2)** DISCO has around 0.6dB PSNR advantage over all existing or baseline methods. **(3)** By introducing 1% data communication, the PSNR can be improved from 31.69 to 32.49.

V. Further Discussions and Comparisons

A. *Number of nodes and other parallelism*

One of the motivations for distributing DNN inference to multiple nodes is to reduce the memory requirement of each node. To demonstrate the benefit of multiple nodes, we denote by "ResNet50/X" the DNN with 1/X of the layer width of a standard ResNet50 model. For example, ResNet50/2 is a ResNet model with 50 layers and each layer has one half channels of the original ResNet50 [39]. The following is assumed: 1) Each node has a memory capacity that is slightly larger than the requirement of inferring the ResNet50/8 model, but not significantly larger ones. 2) The system has comparable communication and computation latency, so reducing either of them will be helpful. By investigating the feature size and computation FLOPs of a sequence of ResNet50/X models, the communication bandwidth $B = 37.5$MB/s and computation speed $C = 3.75$GOP/s are reasonable configurations to fulfill the above assumption in this subsection. The overall inference latency is then calculated measuring data communication latency ($\frac{\text{Bytes transmitted}}{B}$) and computation latency ($\frac{\text{FLOPS}}{C}$). See Appendix for details.

If we have N such nodes, we compare two paralleled computing methods in Figure 4. The first is our "within-layer" model parallelism (Model Parl.) and the second is to partition the model into sequential stages of layers and each stage is inferred on one node. This is often referred to as *pipeline parallelism* (Pipeline Parl.). In "Model Parl.", there is a tiny proportion of data communications between pairs of N=2,4,8 nodes ($S_{\text{comm}} \approx 99\%$) such that the memory requirement to handle the extra 1% data does not exceed the memory capacity of each node. In "Pipeline Parl.", a ResNet50/X model is sequentially partitioned into N=2,4,8 nodes and data communication happens between those partitions. The values of "X" are chosen such that each node has the same memory requirement of a ResNet50/8.

Figure 4 shows the comparison and a few phenomena can be observed. 1) Model parallelism by DISCO significantly

TABLE III: Results of SSD300 models on COCO2017.

Method	S_{comm}	S_{comp}	mAP (%)	Comments
Dense [41]	0%	0%	26.0	
Random Sparse Communication (baseline)	80%	40%	23.6	Uniform S_{comm}
	90%	45%	23.2	
	95%	47.5%	22.6	
	100%	50%	19.6	
Dense-then-split [3]	18%	17%	24.7	Split @ Loc. 1
	60%	42%	21.8	Split @ Loc. 2
	81%	47%	20.3	Split @ Loc. 3
Split-then-aggregate [4]	18%	3%	25.5	Aggr. @ Loc. 1
	40%	8%	24.6	Aggr. @ Loc. 2
	82%	33%	21.5	Aggr. @ Loc. 3
DISCO (Ours)	80%	40%	**26.7**	Uniform S_{comm}
	90%	45%	**26.1**	
	95%	47.5%	**25.6**	
	99%	49.5%	**24.6**	

TABLE IV: Results of DeeplabV3+ models on PASCAL VOC2012.

Method	S_{comm}	S_{comp}	mIoU (%)	Comments
Dense [42]	0%	0%	77.2	
Random Sparse Communication (baseline)	80%	40%	74.4	Uniform S_{comm}
	90%	45%	72.8	
	95%	47.5%	70.9	
	99%	49.5%	66.8	
	100%	50%	64.5	
Dense-then-split [3]	26%	27%	76.5	Split @ Loc. 1
	63%	44%	69.0	Split @ Loc. 2
	81%	47%	66.1	Split @ Loc. 3
Split-then-aggregate [4]	34%	6%	77.2	Aggr. @ Loc. 1
	71%	23%	68.8	Aggr. @ Loc. 2
	79%	33%	64.5	Aggr. @ Loc. 3
DISCO (Ours)	80%	40%	**78.5**	Uniform S_{comm}
	90%	45%	**78.1**	
	95%	47.5%	**77.5**	
	99%	49.5%	**76.7**	

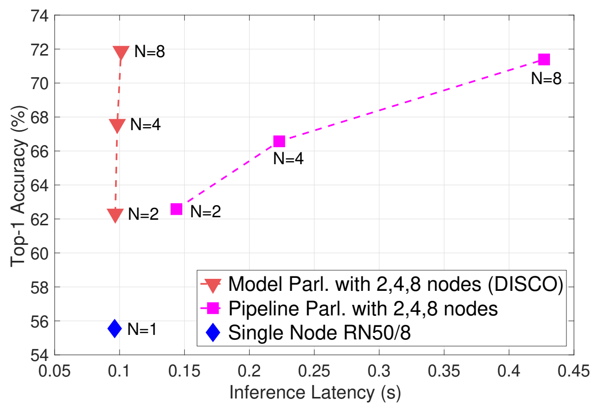

Fig. 4: Comparison of different parallel computing methods.

reduces the inference latency with the same number of nodes. Larger number of nodes has greater latency improvement. This is because the computation of one input is distributed in model parallelism, but not in the pipeline parallelism. 2) With 5% latency overhead, DISCO can significantly improve the accuracy of single-node inference of a small ResNet50/8 model (from 55.55% to 71.89%).

VI. Conclusions

In this paper, we propose a framework, called DISCO, to train DNNs for distributed inference with sparse communications. We convert the problem of selecting the subset of data for transmission to a DNN sparsification problem and show that DISCO can reduce the data communication by 75% with no or negligible accuracy degradation.

References

[1] R. Latha, G. R. R. Sreekanth, R. Suganthe, and R. E. Selvaraj, "A survey on the applications of deep neural networks," in *2021 International Conference on Computer Communication and Informatics (ICCCI)*, 2021.

[2] Wikipedia contributors, "ESP32 — Wikipedia, the free encyclopedia," 2022, [Online; accessed 6-Feb-2022]. [Online]. Available: https://en.wikipedia.org/wiki/ESP32

[3] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization," in *Proceedings of the 34th International Conference on Machine Learning*, 2017, pp. 1866–1874.

[4] X. Dong, Z. Li, M. Li, Z. Qu, B. D. Salvo, C. Liu, and H.-T. Kung, "SplitNets: Designing neural architectures for efficient distributed computing on head-mounted systems," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[5] R. Hadidi, B. Asgari, J. Cao, Y. Bae, D. E. Shim, H. Kim, S.-K. Lim, M. S. Ryoo, and H. Kim, "LCP: A low-communication parallelization method for fast neural network inference in image recognition," *arXiv preprint arXiv:2003.06464*, 2020.

TABLE V: Results of ESRGAN models on DIV2K.

Method	S_{comm}	S_{comp}	PSNR (dB)	Comments
Dense [43]	0%	0%	32.56	
Random Sparse Communication (baseline)	80%	40%	31.82	Uniform S_{comm}
	90%	45%	31.76	
	95%	47.5%	31.75	
	100%	50%	31.69	
Dense-then-split [3]	20%	10%	31.99	Split @ Loc. 1
	44%	22%	31.91	Split @ Loc. 2
	69%	34%	31.87	Split @ Loc. 3
Split-then-aggregate [4]	24%	12%	31.94	Aggr. @ Loc. 1
	48%	24%	31.96	Aggr. @ Loc. 2
	73%	36%	31.85	Aggr. @ Loc. 3
DISCO (Ours)	60%	30%	**32.56**	Uniform S_{comm}
	80%	40%	**32.47**	
	90%	45%	**32.46**	
	95%	47.5%	**32.41**	
	99%	49.5%	**32.49**	

[6] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. a. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, vol. 25, 2012.

[7] X. Li, G. Zhang, K. Li, and W. Zheng, "Deep learning and its parallelization," in *Big Data*, R. Buyya, R. N. Calheiros, and A. V. Dastjerdi, Eds., 2016, pp. 95–118.

[8] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, 2021.

[9] H.-J. Jeong, I. Jeong, H.-J. Lee, and S.-M. Moon, "Computation offloading for machine learning web apps in the edge server environment," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, 2018, pp. 1492–1499.

[10] 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, pp. 615–629, 04 2017.

[11] D. J. Pagliari, R. Chiaro, E. Macii, and M. Poncino, "Crime: Input-dependent collaborative inference for recurrent neural networks," *IEEE Transactions on Computers*, vol. 70, no. 10, pp. 1626–1639, 2021.

[12] D. Hu and B. Krishnamachari, "Fast and accurate streaming cnn inference via communication compression on the edge," in *2020 IEEE/ACM Fifth International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2020, pp. 157–163.

[13] S. Jung and H.-W. Lee, "Optimization framework for splitting DNN inference jobs over computing networks," 2021.

[14] A. Parthasarathy and B. Krishnamachari, "Defer: Distributed edge inference for deep neural networks," in *2022 14th International Conference on COMmunication Systems and NETworkS (COMSNETS)*, 2022, pp. 749–753.

[15] R. Stahl, A. Hoffman, D. Mueller-Gritschneder, A. Gerstlauer, and U. Schlichtmann, "Deeperthings: Fully distributed CNN inference on resource-constrained edge devices," *International Journal of Parallel Programming*, 2021.

[16] Z. Zhao, K. M. Barijough, and A. Gerstlauer, "Deepthings: Distributed adaptive deep learning inference on resource-constrained iot edge clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[17] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 328–339.

[18] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy, "MCDNN: An approximation-based execution framework for deep stream processing under resource constraints," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*. Association for Computing Machinery, 2016, p. 123–136.

[19] Y. Matsubara, S. Baidya, D. Callegaro, M. Levorato, and S. Singh, "Distilled split deep neural networks for edge-assisted real-time systems," in *Proceedings of the 2019 Workshop on Hot Topics in Video Analytics and Intelligent Edges*. Association for Computing Machinery, 2019, p. 21–26.

[20] T. Mohammed, C. Joe-Wong, R. Babbar, and M. D. Francesco, "Distributed inference acceleration with adaptive DNN partitioning and offloading," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, 2020, pp. 854–863.

[21] Q. Li, L. Huang, Z. Tong, T.-T. Du, J. Zhang, and S.-C. Wang, "Dissec: A distributed deep neural network inference scheduling strategy for edge clusters," *Neurocomputing*, vol. 500, pp. 449–460, 2022.

[22] C. Hu and B. Li, "Distributed inference with deep learning models across heterogeneous edge devices," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 330–339.

[23] X. Hou, Y. Guan, T. Han, and N. Zhang, "Distredge: Speeding up convolutional neural network inference on distributed edge devices," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2022, pp. 1097–1107.

[24] A. L. Gaunt, M. A. Johnson, M. Riechert, D. Tarlow, R. Tomioka, D. Vytiniotis, and S. Webster, "AMPNet: Asynchronous model-parallel training for dynamic neural networks," *arXiv preprint arXiv:1705.09786*, 2017.

[25] J. Giacomoni, T. Moseley, and M. Vachharajani, "Fastforward for efficient pipeline parallelism: A cache-optimized concurrent lock-free queue." Association for Computing Machinery, 2008, p. 43–52.

[26] M. I. Gordon, W. Thies, and S. P. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," in *ASPLOS XII*, 2006.

[27] L. Guan, W. Yin, D. Li, and X. Lu, "XPipe: Efficient pipeline model parallelism for multi-GPU DNN training," *arXiv preprint arXiv:1911.04610*, 2019.

[28] A. Harlap, D. Narayanan, A. Phanishayee, V. Seshadri, N. Devanur, G. Ganger, and P. Gibbons, "Pipedream: Fast and efficient pipeline parallel DNN training," *arXiv preprint arXiv:1806.03377*, 2018.

[29] C.-C. Chen, C.-L. Yang, and H.-Y. Cheng, "Efficient and robust parallel dnn training through model parallelism on multi-gpu platform," *arXiv preprint arXiv:1809.02839*, 2018.

[30] M. Tanaka, K. Taura, T. Hanawa, and K. Torisawa, "Automatic graph partitioning for very large-scale deep learning," *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pp. 1004–1013, 2021.

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

[32] Z. Jia, S. Lin, C. R. Qi, and A. Aiken, "Exploring hidden dimensions in parallelizing convolutional neural networks," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, vol. 80, 2018, pp. 2279–2288.

[33] M. Wang, C.-c. Huang, and J. Li, "Supporting very large models using automatic dataflow graph partitioning," in *Proceedings of the Fourteenth EuroSys Conference 2019*. Association for Computing Machinery, 2019.

[34] L. Abrahamyan, Y. Chen, G. Bekoulis, and N. Deligiannis, "Learned gradient compression for distributed deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[35] A. F. Aji and K. Heafield, "Sparse communication for distributed gradient descent," in *EMNLP*, 2017.

[36] N. Ström, "Scalable distributed DNN training using commodity gpu cloud computing," in *Interspeech 2015*, 2015.

[37] D. Alistarh, D. Grubic, J. Li, R. Tomioka, and M. Vojnovic, "QSGD: Communication-efficient SGD via gradient quantization and encoding," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[38] A. Abdi and F. Fekri, "Nested dithered quantization for communication reduction in distributed training," *arXiv preprint arXiv:1904.01197*, 2019.

[39] 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 (CVPR)*, 2016.

[40] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou, "Training data-efficient image transformers and distillation through attention," in *International Conference on Machine Learning*, vol. 139, July 2021, pp. 10 347–10 357.

[41] NVIDIA, "SSD300 v1.1 For PyTorch," https://github.com/NVIDIA/DeepLearningExamples/tree/master/PyTorch/Detection/SSD, 2020.

[42] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, 2018.

[43] X. Wang, K. Yu, S. Wu, J. Gu, Y. Liu, C. Dong, Y. Qiao, and C. C. Loy, "ESRGAN: Enhanced super-resolution generative adversarial networks," in *The European Conference on Computer Vision Workshops (ECCVW)*, September 2018.

[44] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[45] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020.

[46] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[47] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *Proceedings of the European conference on computer vision (ECCV)*. Springer, 2016, pp. 21–37.

[48] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, P. P. James Hays, D. Ramanan, C. L. Zitnick, and P. Dollár, "COCO 2017." [Online]. Available: http://cocodataset.org/

[49] M. Everingham and J. Winn, "The PASCAL visual object classes challenge 2012 (VOC2012) development kit," *Pattern Analysis, Statistical Modelling and Computational Learning, Tech. Rep*, vol. 8, p. 5, 2011.

[50] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.