Lecture 09: Distributed Machine Learning for Training and Inference

Some Notes

- Project proposals have been graded, and feedback has been posted.
- Lab 2 grades will be released this weekend.
- Lab 3 is due in two weeks.
- We'll also hold a round of project meetings to check on each team's progress.
- Change on Course Schedule

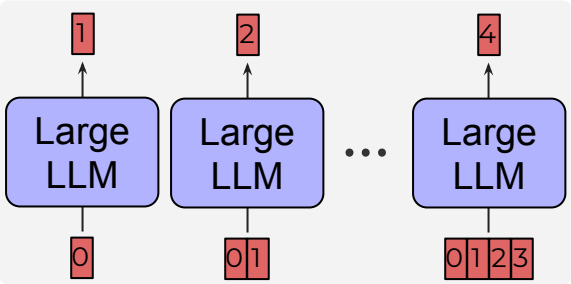
Recap

- Efficient training of DNNs
  - Efficient computing
  - Efficient storage
- Parameter efficient finetuning
- Speculative Decoding

Topics

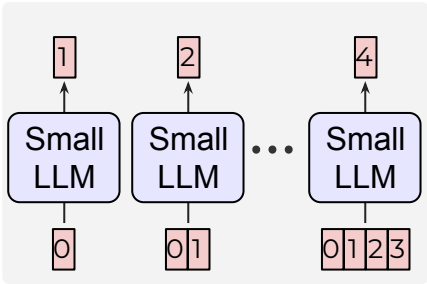
- Speculative Decoding (Continue)
- Distributed DNN Training
- Distributed DNN Inference
- Federated Learning

Speculative Decoding



Accurate but slow

$T_{tot} = NT_{p,1}$



Fast but inaccurate

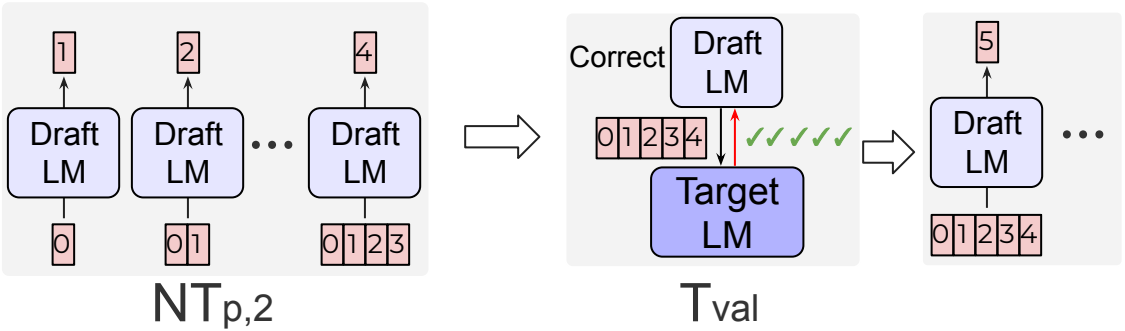
$T_{tot} = NT_{p,2}$

- Speculative decoding enables lossless token generation with low latency.

NYU SAI LAB

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." *International Conference on Machine Learning*. PMLR, 2023.

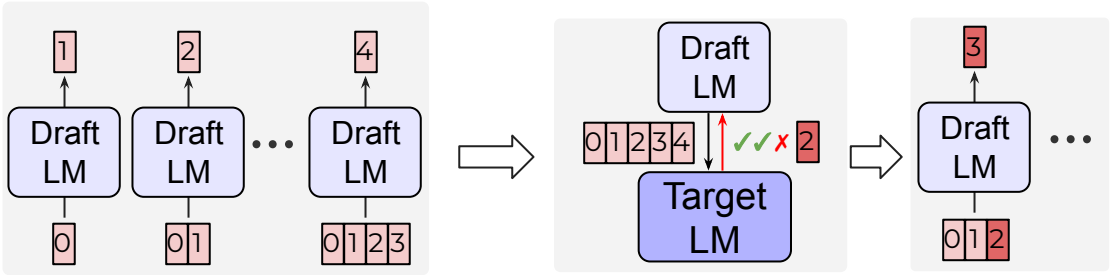
Speculative Decoding



$$T_{tot} = NT_{p,2} + T_{val} < NT_{p,1}$$

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." *International Conference on Machine Learning*. PMLR, 2023.

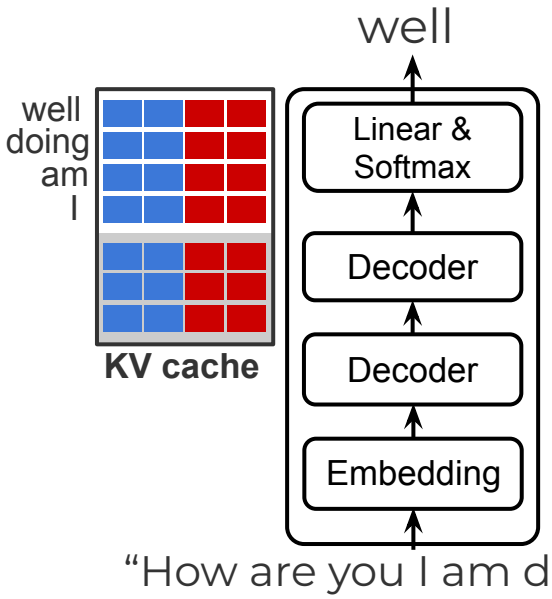
Speculative Decoding

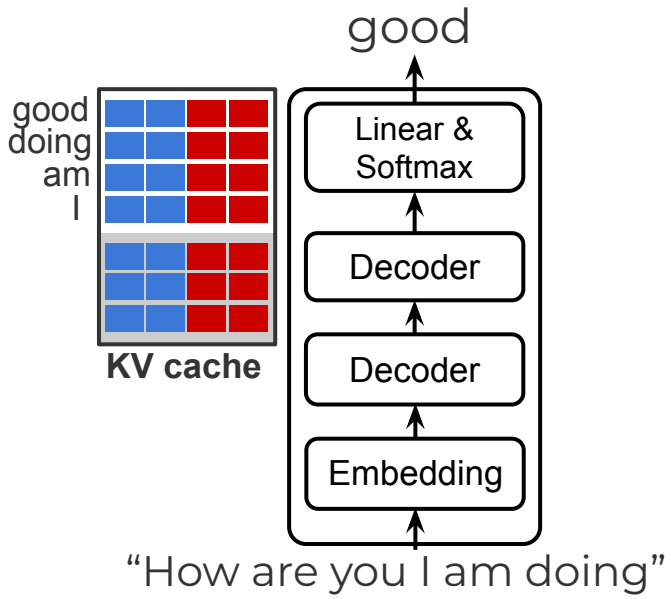


- If the token is incorrect, the target model provides the correct token to the draft model to help it generate subsequent tokens more accurately.
- If the amount of tokens that pass the verification is too low, then it is possible that speculative decoding is slower than autoregressive baseline.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." *International Conference on Machine Learning*. PMLR, 2023.

LLM Decoding





- We can simply select the token with the highest score. But better results are achieved if the model considers other words as well. So a better strategy is to sample a word from the entire list using the score as the probability of selecting that word.

Speculative Decoding

- To increase the diversity of the LLM output, a better strategy is to sample a word from the entire list using the score as the probability of selecting that word.
- Let p(x), q(x) denote the probability density function specified by the target and draft LLM
- To sample x ~ p(x), we instead sample x ~ q(x), keeping it if $q(x) \leq p(x)$, and in case $q(x) > p(x)$ we reject the sample with probability $1 - p(x)/q(x)$ and sample x again from an adjusted distribution $p'(x) = \text{norm}(\max(0, p(x) - q(x)))$ instead.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias. "Fast inference from transformers via speculative decoding." *International Conference on Machine Learning*. PMLR, 2023.

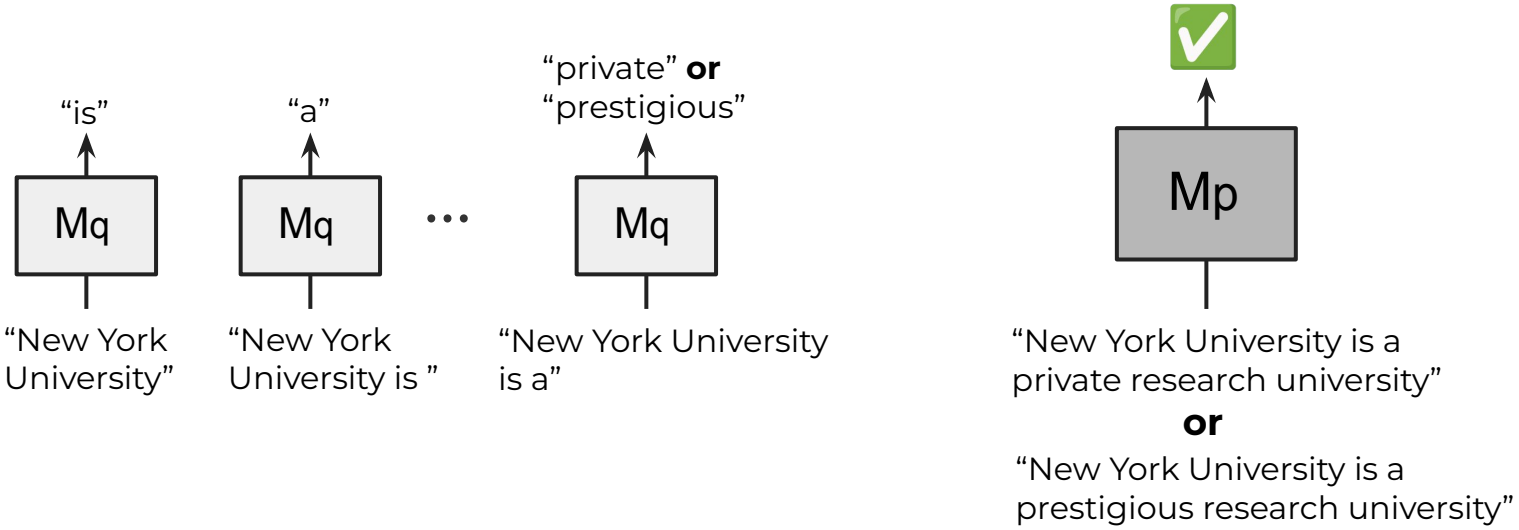
Speculative Decoding

- Speculative decoding does not save computation, but greatly reduce the memory traffic by reducing the number of memory reads, further reducing the **overall latency**.

Algorithm 1 SpeculativeDecodingStep

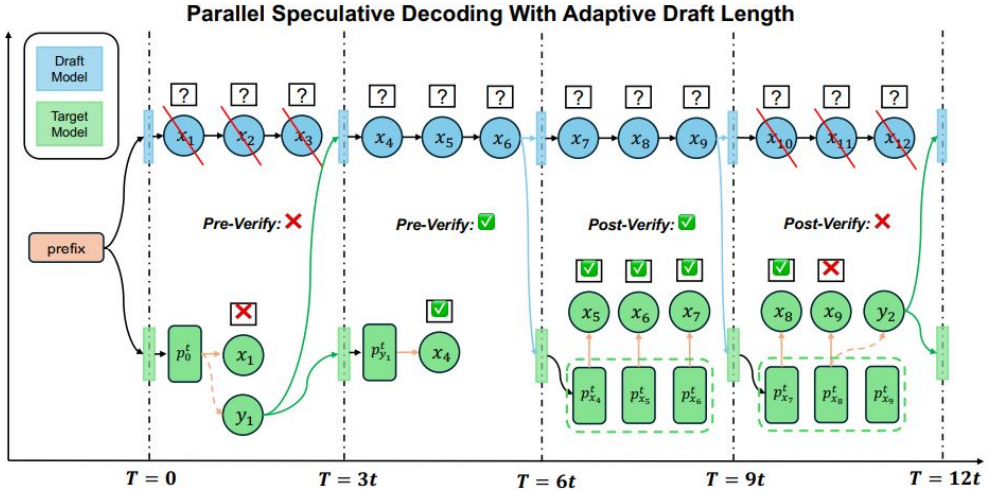
Inputs: $M_p, M_q, prefix$.
▷ Sample γ guesses $x_{1,\ldots,\gamma}$ from M_q autoregressively.
for $i = 1$ **to** γ **do**
  $q_i(x) \leftarrow M_q(prefix + [x_1, \ldots, x_{i-1}])$
  $x_i \sim q_i(x)$
end for
▷ Run M_p in parallel.
$p_1(x), \ldots, p_{\gamma+1}(x) \leftarrow$
  $M_p(prefix), \ldots, M_p(prefix + [x_1, \ldots, x_\gamma])$
▷ Determine the number of accepted guesses n.
$r_1 \sim U(0,1), \ldots, r_\gamma \sim U(0,1)$
$n \leftarrow \min(\{i - 1 \mid 1 \leq i \leq \gamma, r_i > \frac{p_i(x)}{q_i(x)}\} \cup \{\gamma\})$
▷ Adjust the distribution from M_p if needed.
$p'(x) \leftarrow p_{n+1}(x)$
if $n < \gamma$ **then**
  $p'(x) \leftarrow norm(max(0, p_{n+1}(x) - q_{n+1}(x)))$
end if
▷ Return one token from M_p, and n tokens from M_q.
$t \sim p'(x)$
return $prefix + [x_1, \ldots, x_n, t]$

SpecInfer



Miao, Xupeng, et al. "SpecInfer: Accelerating Generative Large Language Model Serving with Tree-based Speculative Inference and Verification." *arXiv preprint arXiv:2305.09781* (2023).

Parallel Speculative Decoding

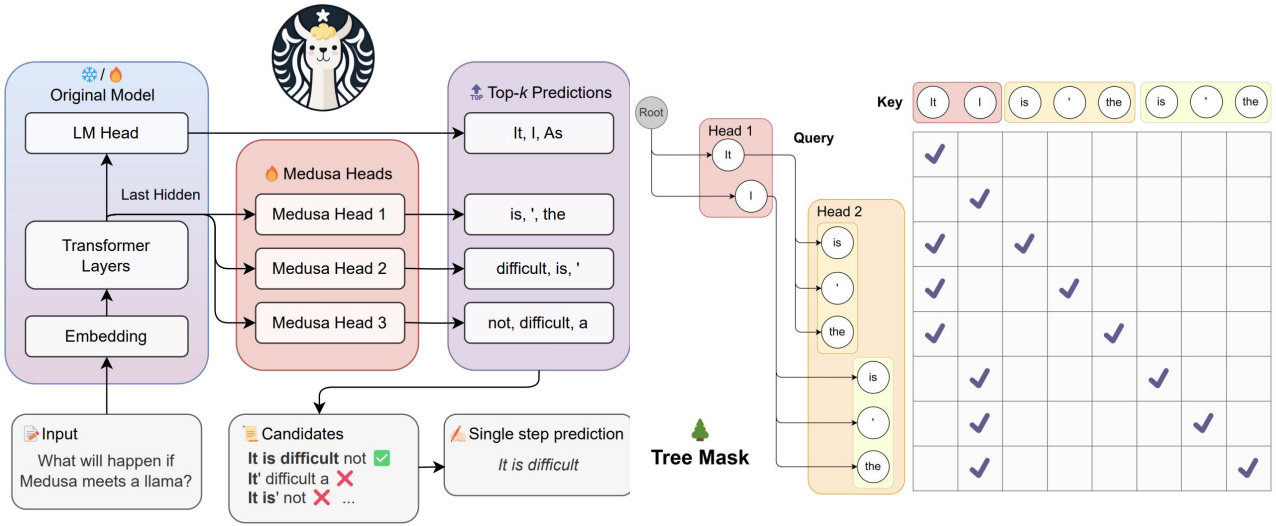


- PEARL is a parallel inference framework based on speculative decoding which utilizes pre-verify and post-verify to achieve adaptive draft length.
- The draft model continues to decode during the verification stage.
- If the verification fails, the windows size will become 1 in the next cycle.

NYU SAI LAB

Liu, Tianyu, Yun Li, Qitan Lv, Kai Liu, Jianchen Zhu, and Winston Hu. "Parallel speculative decoding with adaptive draft length." arXiv preprint arXiv:2408.11850 (2024).

Medusa



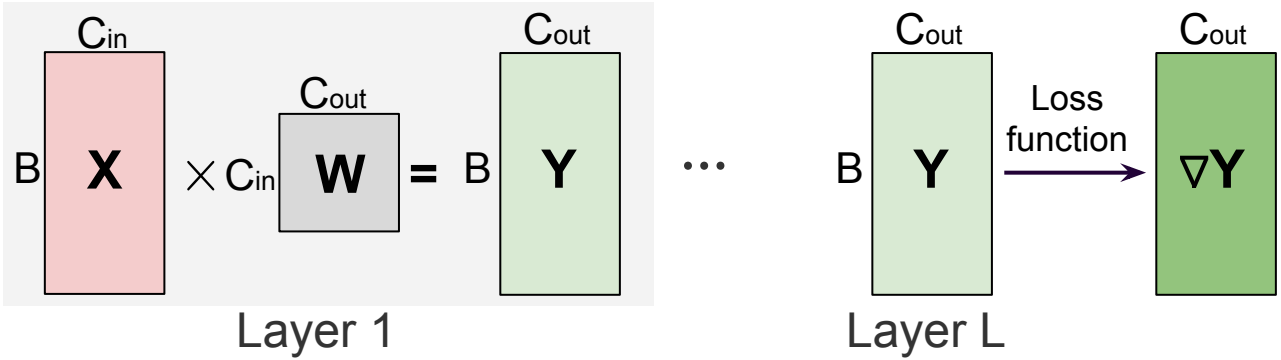
- Adding extra decoding heads to predict multiple subsequent tokens in parallel.

Cai, Tianle, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. "Medusa: Simple llm inference acceleration framework with multiple decoding heads." arXiv preprint arXiv:2401.10774 (2024).

Topics

- Speculative Decoding (Continue)
- Distributed DNN Training
- Distributed DNN Inference
- Federated Learning

Forward Pass for Linear Layer



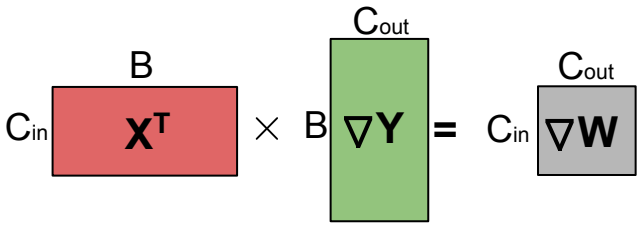
B: batch size C_{in}: input channels C_{out}: output channels

X: input maps **W**: weight filters **Y**: output maps

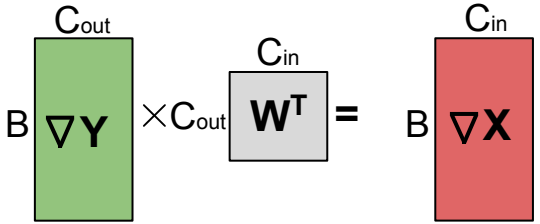
- The fully-connected layer during the forward propagation can be converted into matrix multiplications.

Backward Pass for Linear Layer

Weight Gradient Computation



Data Gradient Computation



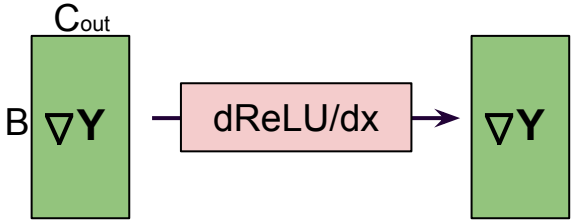
X: input maps **W**: weight filters **Y**: output maps

∇**X**: input gradient ∇**W**: weight gradient ∇**Y**: output gradient

- DNN backward propagation involves two matrix multiplications

Backward Pass for Linear Layer

Data Gradient Computations



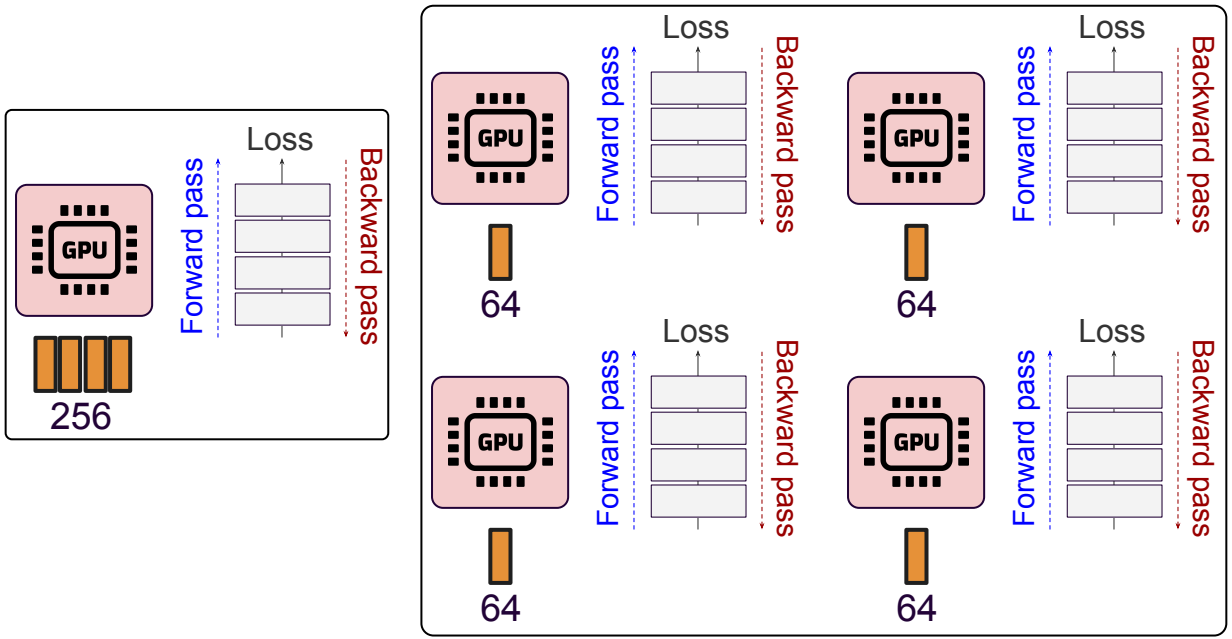
Weight Gradient Updates

X: input maps **W**: weight filters **Y**: output maps

∇**X**: input gradient ∇**W**: weight gradient ∇**Y**: output gradient

- DNN backward propagation involves two matrix multiplications

Two Types of Parallelism



- Data is partitioned and allocated equally among the GPUs.
- Each GPU will process independently.

Distributed DNN Training: Data Parallelism

- To train DNN in a distributed fashion, we need to batchify the training datasets.
- Assume a batch size of $b \in B$, x denotes a batch of training dataset.
- Let η represent the learning rate. w_t represents the weight at t.
- We assume that the data will be distributed in an independent and identically distributed (IID) fashion.

$$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w)$$

Loss function

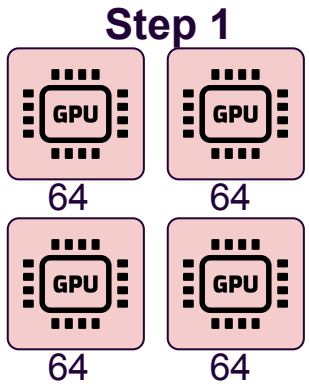
$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

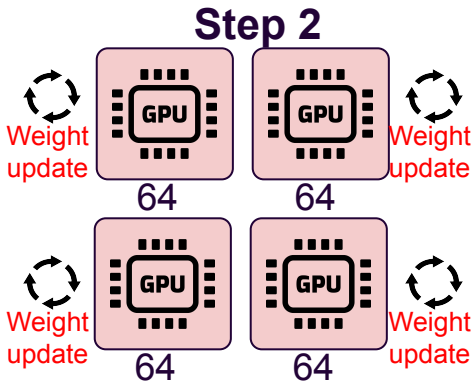
Weight update

NYU SAI LAB

Parameter Server

- A parameter server is a distributed system used to manage and synchronize the parameters (weights) of a machine learning model during training, especially in large-scale and distributed training scenarios.

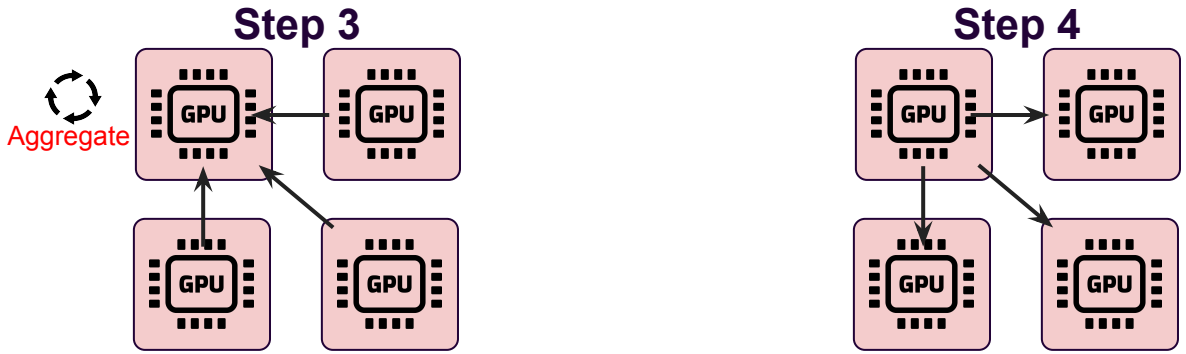




A total batch size of 256

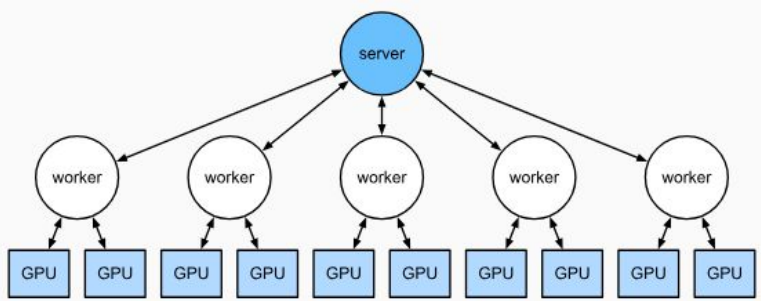
Li, Mu, et al. "Scaling distributed machine learning with the parameter server." *11th USENIX Symposium on operating systems design and implementation (OSDI 14)*. 2014.

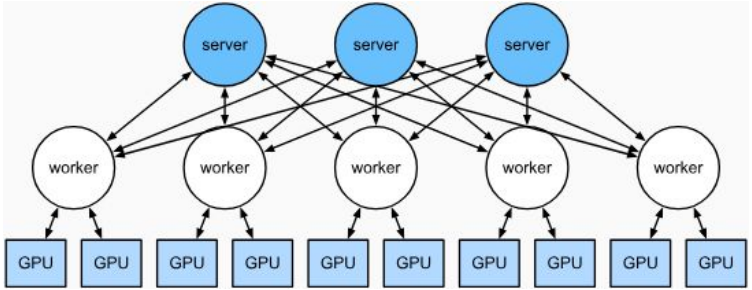
Parameter Server



- Total amount of communication: 2(N-1)G.
- N is the number of nodes, G is the size of the weight gradient.
- If a worker node fails, other nodes can continue training without significant disruption. But PS scheme is not scalable, the central node can not handle all the servers, as the number of nodes increases.

Parameter Server



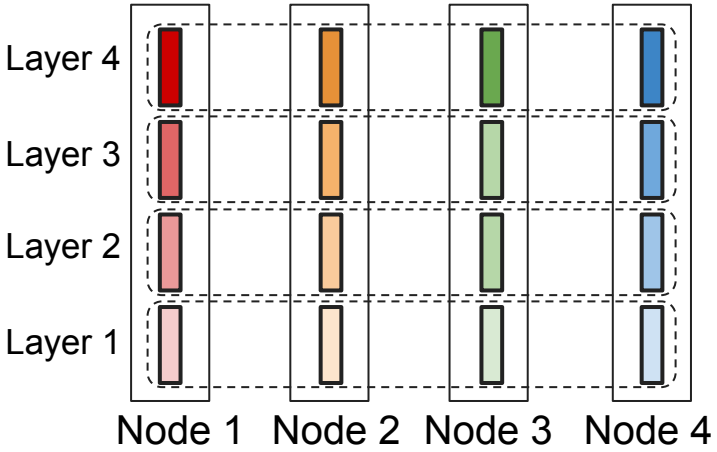


- To improve performance, we can deploy multiple parameter servers as backup nodes
- However, this approach introduces additional computation and communication overhead.

All Reduce

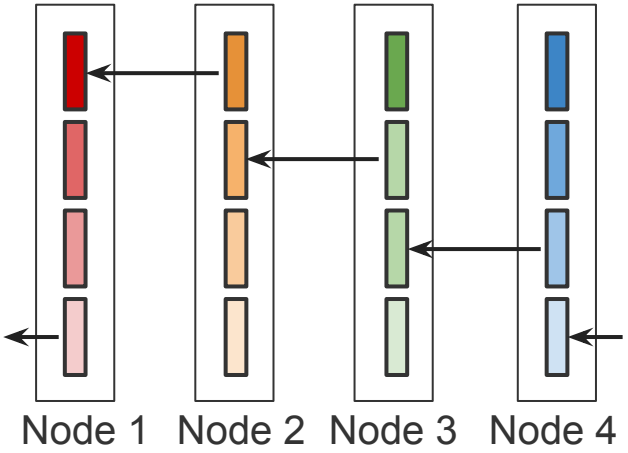
- All-reduce is a communication operation widely used in distributed deep neural network (DNN) training to synchronize and aggregate data across multiple computing nodes or devices.
- Detailed training steps:
  - **Forward Pass:** Each node (e.g., GPU) computes the forward pass of the neural network independently using its local mini-batch of data.
  - **Backward Pass:** Each node computes the gradients of the loss with respect to the model parameters.
  - **All-Reduce Step:** The gradients from all nodes are summed together using the all-reduce operation. This summed gradient is then broadcast to all nodes.
  - **Parameter Update:** Each node updates its local copy of the model parameters using the aggregated gradients.

Ring All-Reduce



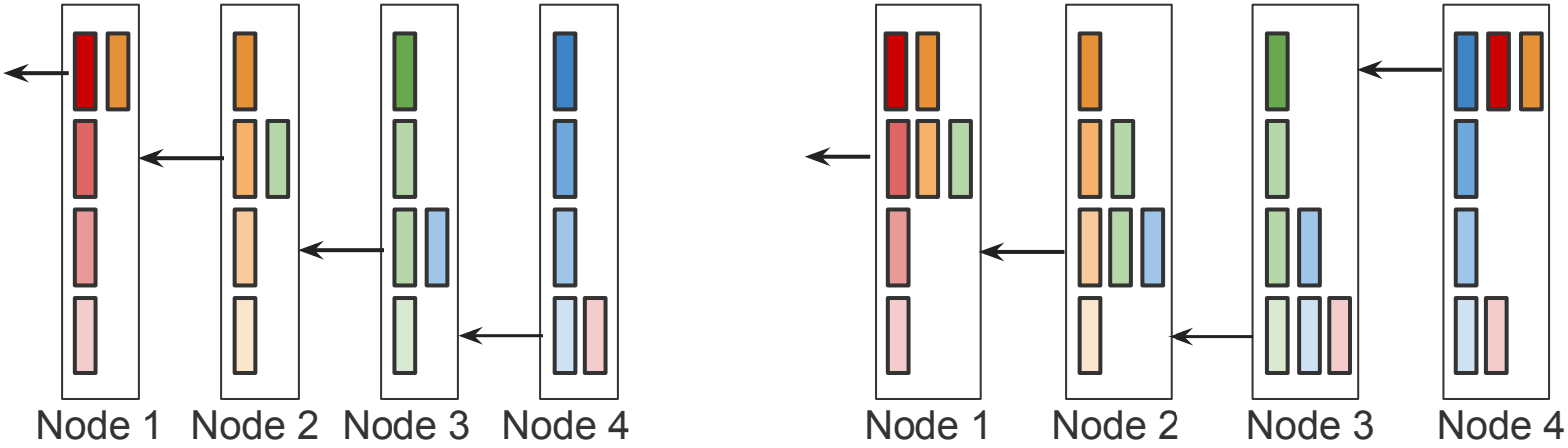
- Assume a neural network with four layers.
- Each node has been assigned with an equivalent amount of training dataset.
- All the nodes have finished training using their own local training dataset.

Ring All-Reduce



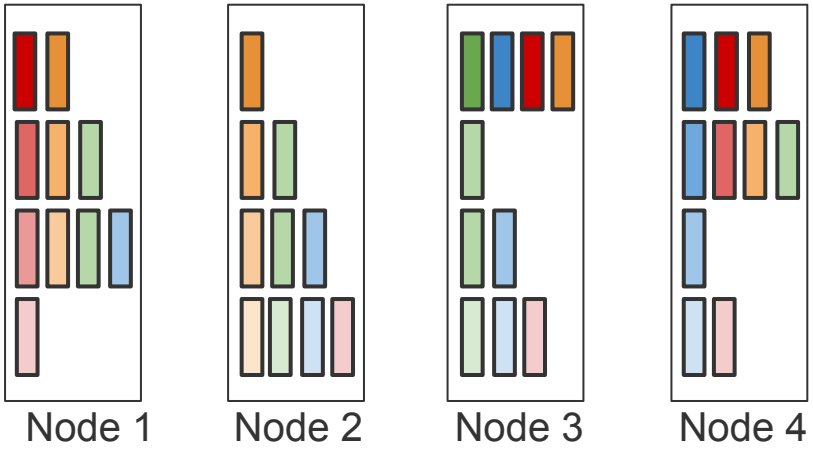
- Nodes are arranged in a ring topology, and each node passes a portion of its data to its neighbor in a circular fashion. This continues until all nodes have the complete reduced data.
- Each node has identical amounts of workload.

Ring All-Reduce



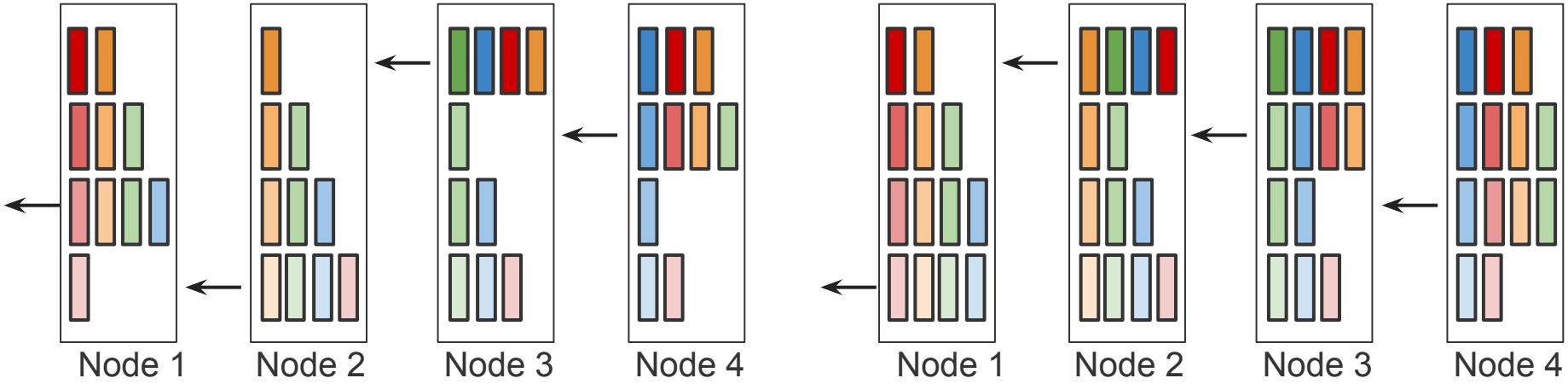
NYU SAI LAB

Ring All-Reduce

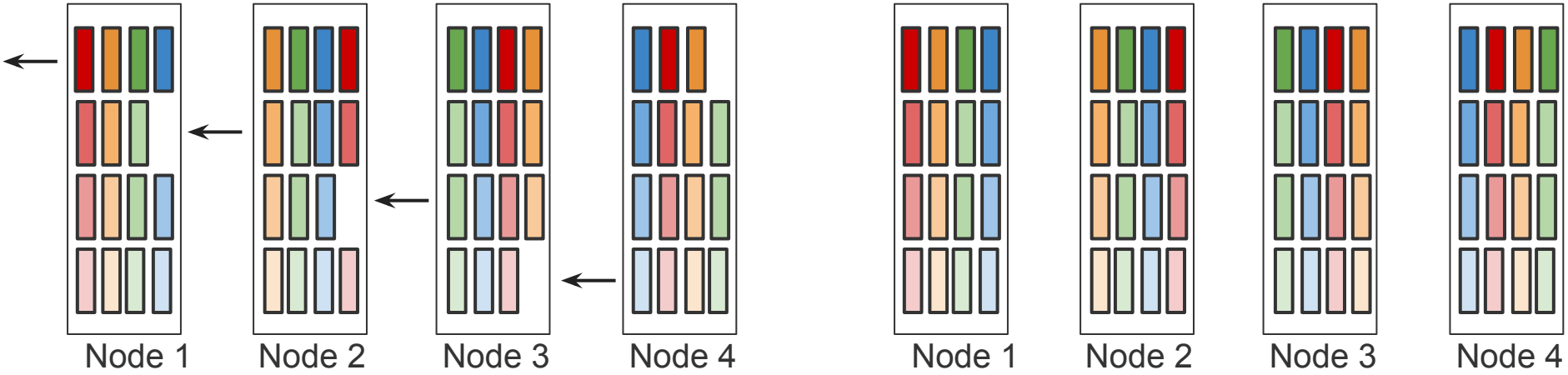


- The end of share-reduce phase.

Ring All-Reduce



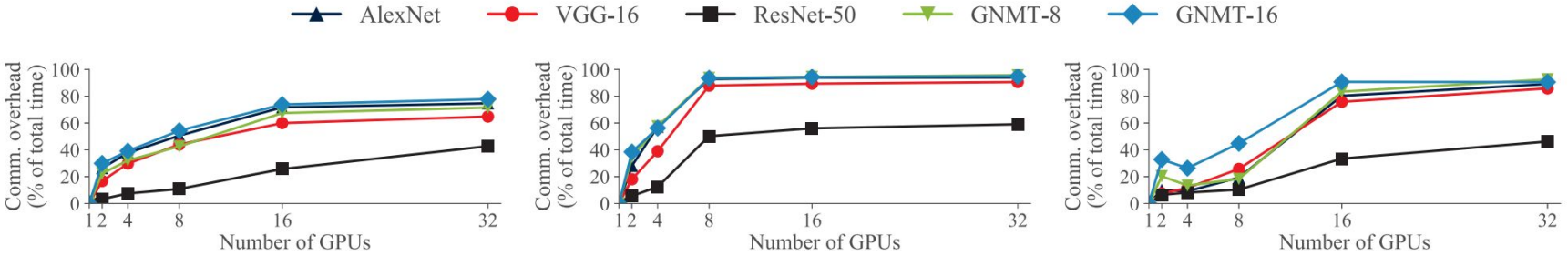
Ring All-Reduce



- Total amount of communication: 2(N-1)G.
- N is the number of nodes, G is the size of the weight gradient.

NYU SAI LAB

Model Parallelism: PipeDream



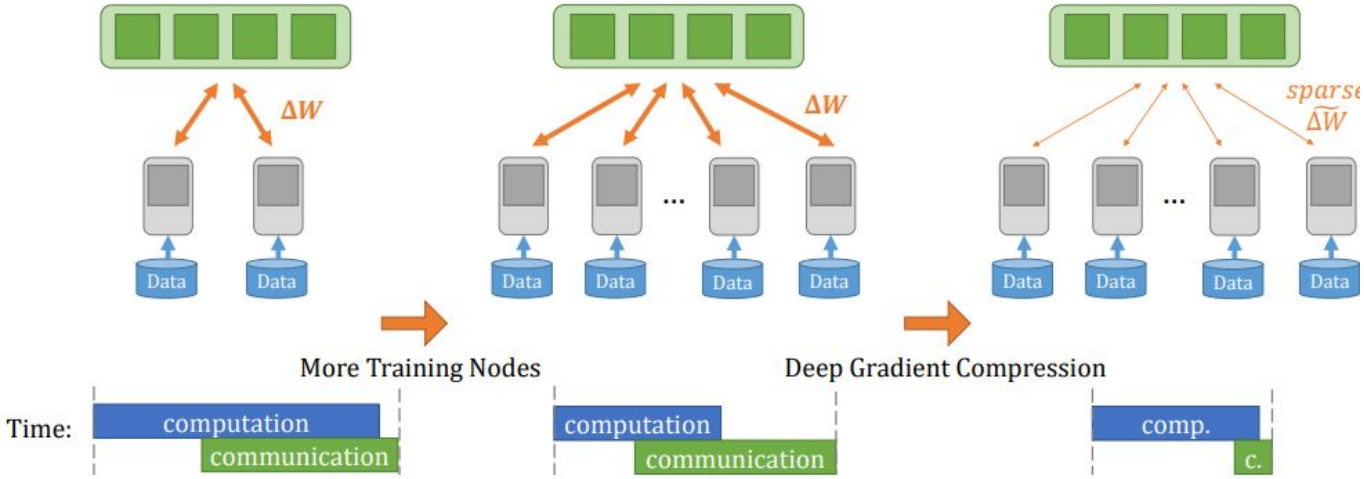
(a) Instances with 8 1080Tis (private cluster). (b) Instances with 4 V100s (Azure). (c) Instances with 8 V100s and NVLink (EC2).

- The communication overhead for many of these models is high despite using multi-GPU servers.
- Applications distributed across multi-GPU servers are bottlenecked by slower inter-server links, as evidenced by communication overheads spiking and then plateauing when training scales out to multiple servers.

NYU SAI LAB

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

Communication Reduction for Distributed Training



- We reduce the communication bandwidth by sending only the important gradients (magnitude > thres).
- The accumulated weight gradient of each layer is transmitted only when its value is larger than a threshold.

NYU SAI LAB

Lin, Yujun, et al. "Deep gradient compression: Reducing the communication bandwidth for distributed training." *arXiv preprint arXiv:1712.01887* (2017).

Communication Reduction for Distributed Training

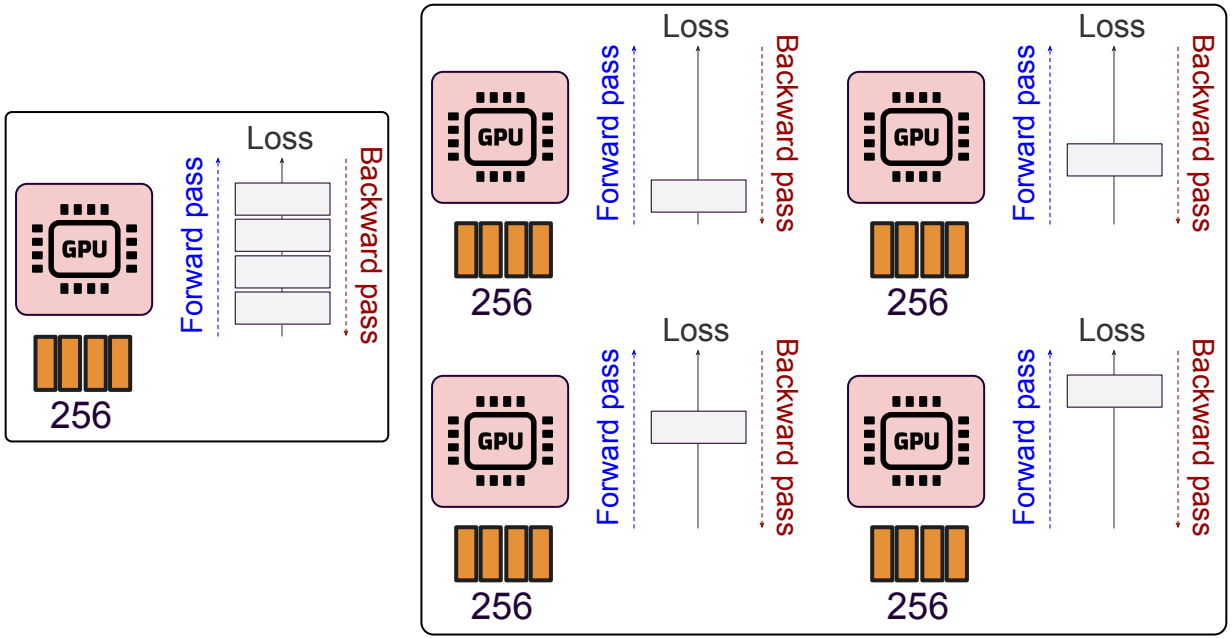
- The gradient is collected locally, only gradient with high magnitude are sent to the central server for model updating.
- Run-length encoding is utilized to compress the sparse gradient.

Algorithm 1 Gradient Sparsification on node k

Input: dataset χ
Input: minibatch size b per node
Input: the number of nodes N
Input: optimization function SGD
Input: init parameters $w = \{w[0], w[1], \cdots, w[M]\}$

$G^k \leftarrow 0$
for $t = 0, 1, \cdots$ **do**
  $G_t^k \leftarrow G_{t-1}^k$
  for $i = 1, \cdots, b$ **do**
    Sample data x from χ
    $G_t^k \leftarrow G_t^k + \frac{1}{Nb} \nabla f(x; w_t)$
  end for
  for $j = 0, \cdots, M$ **do**
    Select threshold: $thr \leftarrow s\%$ of $\left|G_t^k[j]\right|$
    $Mask \leftarrow \left|G_t^k[j]\right| > thr$
    $\widetilde{G}_t^k[j] \leftarrow G_t^k[j] \odot Mask$
    $G_t^k[j] \leftarrow G_t^k[j] \odot \neg Mask$
  end for
  All-reduce $G_t^k : G_t \leftarrow \sum_{k=1}^{N} encode(\widetilde{G}_t^k)$
  $w_{t+1} \leftarrow SGD(w_t, G_t)$
end for

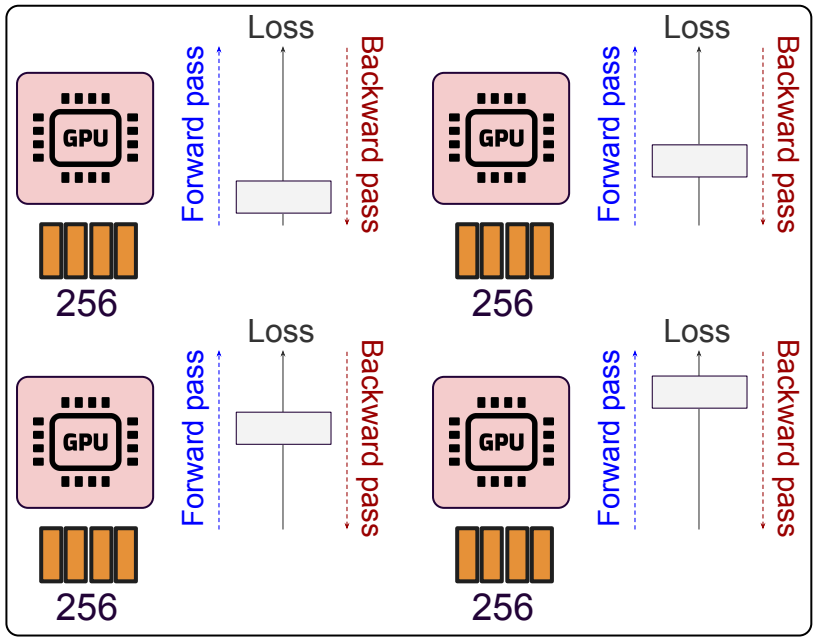
Two Types of Parallelism



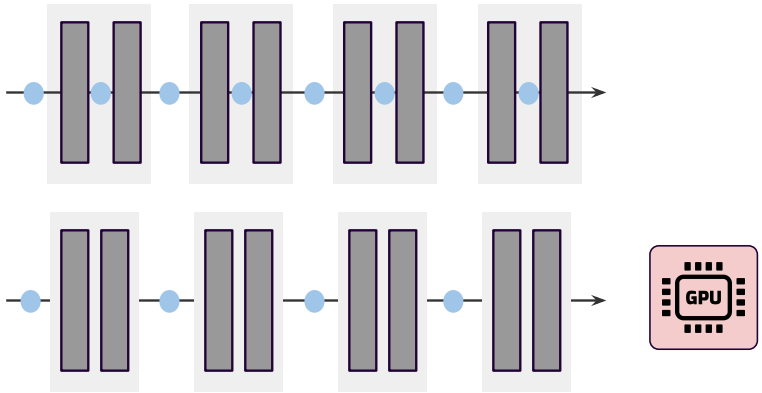
- Data is partitioned and allocated equally among the GPUs.
- Each GPU will process independently.

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *Advances in neural information processing systems* 32 (2019).

Distributed DNN Training: Model Parallelism



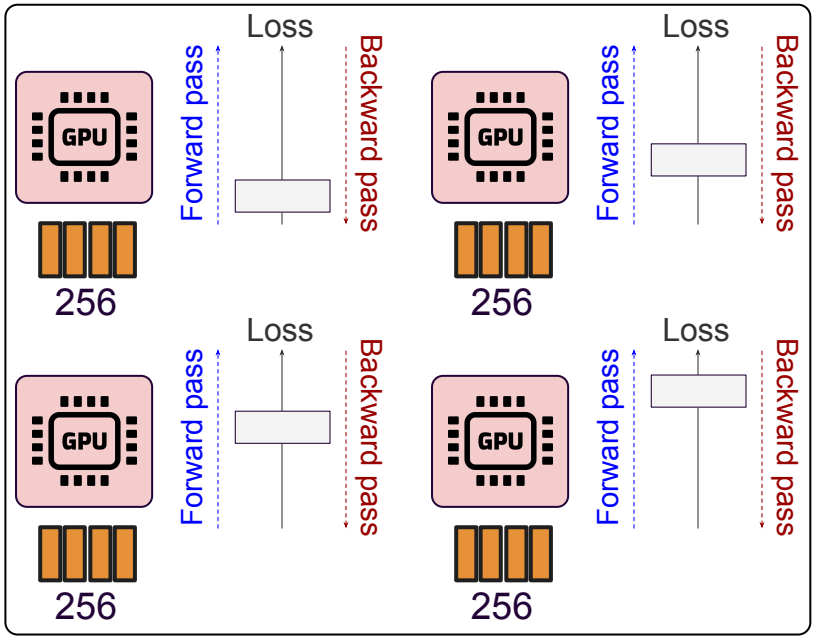
- The peak memory also becomes smaller by N times, where N is the number of devices.

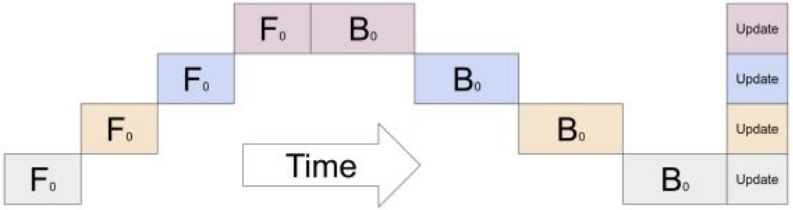


- The peak memory usage can be further reduced by buffering only the inputs of each layer block and recomputing the intermediate results when needed.

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *Advances in neural information processing systems* 32 (2019).

Underutilization During Training

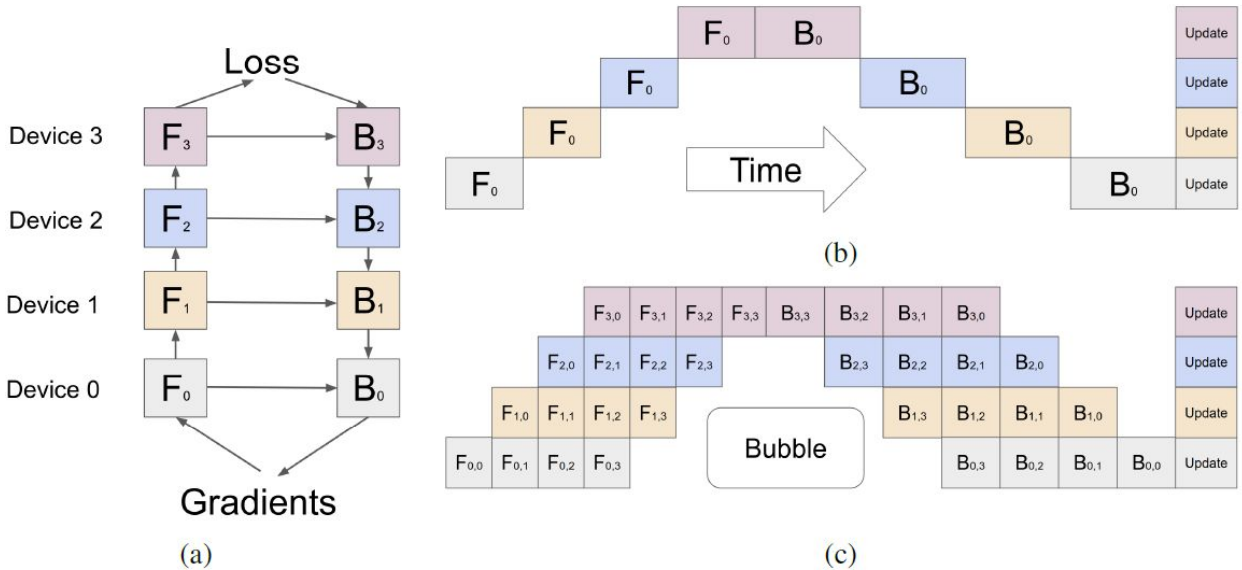




- GPU resources are often underutilized during training, primarily due to the overhead and inefficiencies introduced by model parallelism in the DNN training process.

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *Advances in neural information processing systems* 32 (2019).

Distributed DNN Training: Model Parallelism



- The naive model parallelism strategy leads to severe under-utilization due to the sequential dependency of the network.
- GPipe first divides every mini-batch of size N into M equal micro-batches, enabling different accelerators to work on different micro-batches simultaneously.

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *Advances in neural information processing systems* 32 (2019).

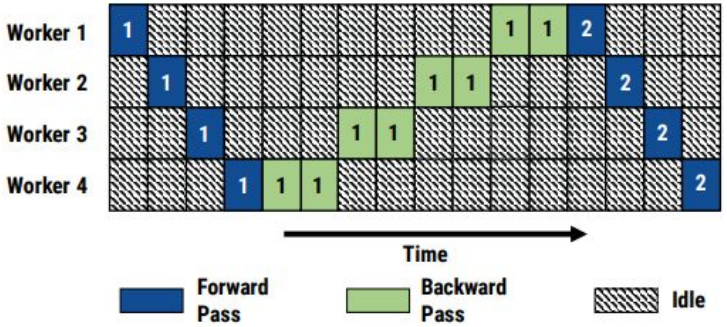
Evaluation Results

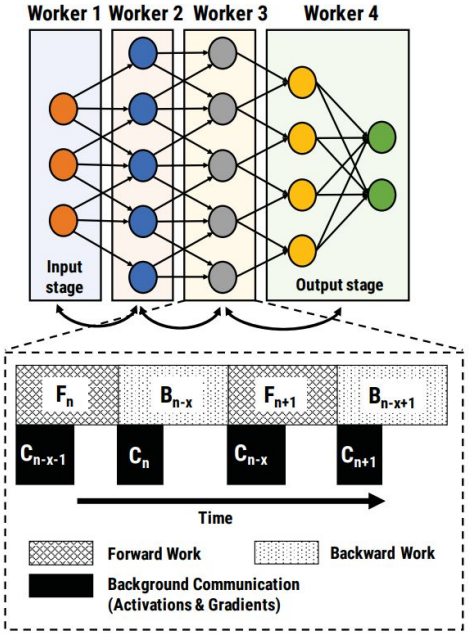
TPU	AmoebaNet			Transformer		
$K =$	2	4	8	2	4	8
$M = 1$	1	1.13	1.38	1	1.07	1.3
$M = 4$	1.07	1.26	1.72	1.7	3.2	4.8
$M = 32$	1.21	1.84	3.48	1.8	3.4	6.3

- Gpipe achieves different levels of acceleration under different number of devices and number of microbatches.

Huang, Yanping, et al. "Gpipe: Efficient training of giant neural networks using pipeline parallelism." *Advances in neural information processing systems* 32 (2019).

Model Parallelism: PipeDream



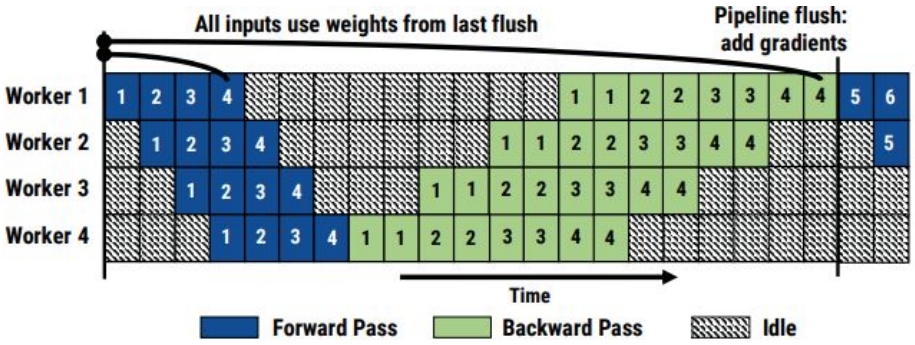


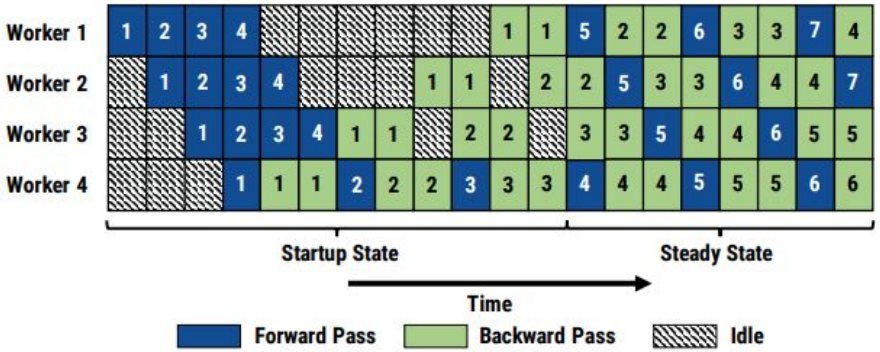
- It will be beneficial to pipeline-parallel assignment and achieve temporal overlap of computation and activation / gradient communication.

NYU SAI LAB

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

Model Parallelism: PipeDream

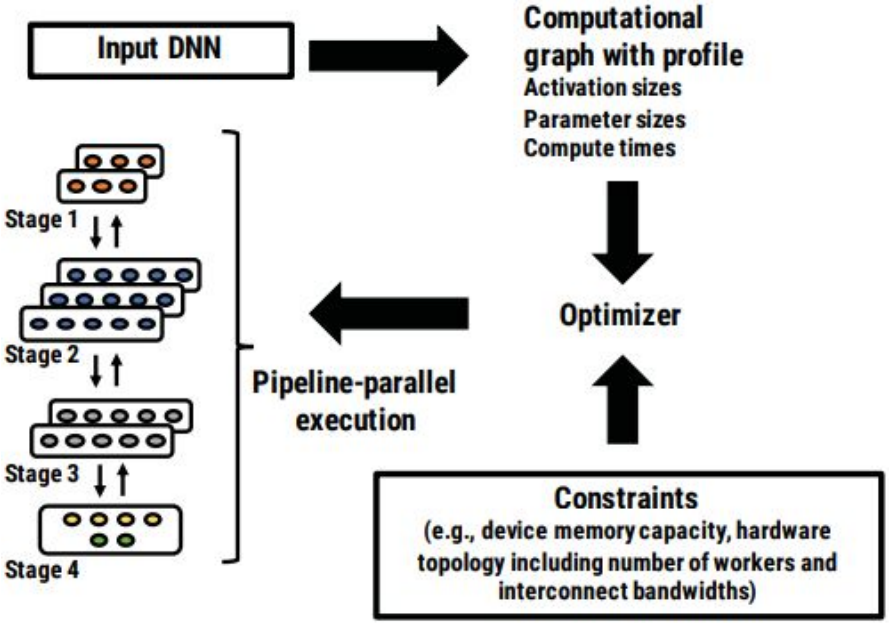




- In this paper, we propose PipeDream, a system that uses Pipeline to enable faster DNN training by combining intra-batch parallelism with inter-batch parallelization.

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

Model Parallelism: PipeDream



- PipeDream's automated mechanism to partition DNN layers into stages.
- PipeDream first profiles the input DNN, to get estimates for each layer's compute time and output size.
- Using these estimates, PipeDream's optimizer partitions layers across available machines, which is then executed by PipeDream's runtime.

NYU SAI LAB

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

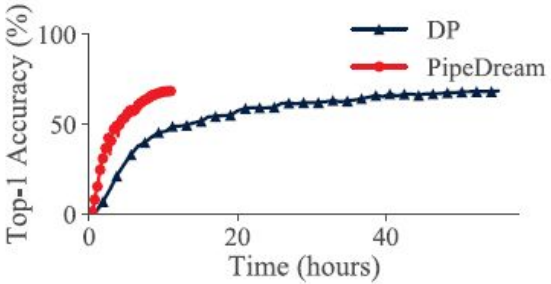
Model Parallelism: PipeDream

Task	Model	Dataset	Accuracy Threshold	# Servers × # GPUs per server (Cluster)	PipeDream Config	Speedup over DP	
						Epoch time	TTA
Image Classification	VGG-16 [48]	ImageNet [44]	68% top-1	4x4 (A)	15-1	5.28×	5.28×
				2x8 (B)	15-1	2.98×	2.46×
	ResNet-50 [26]	ImageNet [44]	75.9% top-1	4x4 (A)	16	1×	1×
				2x8 (B)	16	1×	1×
	AlexNet [37]	Synthetic Data	N/A	4x4 (A)	15-1	4.92×	N/A
				2x8 (B)	15-1	2.04×	N/A
Translation	GNMT-16 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.46×	2.2×
				4x4 (A)	Straight	2.34×	2.92×
				2x8 (B)	Straight	3.14×	3.14×
	GNMT-8 [55]	WMT16 EN-De	21.8 BLEU	1x4 (A)	Straight	1.5×	1.5×
				3x4 (A)	Straight	2.95×	2.95×
				2x8 (B)	16	1×	1×
Language Model	AWD LM [40]	Penn Treebank [41]	98 perplexity	1x4 (A)	Straight	4.25×	4.25×
Video Captioning	S2VT [54]	MSVD [11]	0.294 METEOR	4x1 (C)	2-1-1	3.01×	3.01×

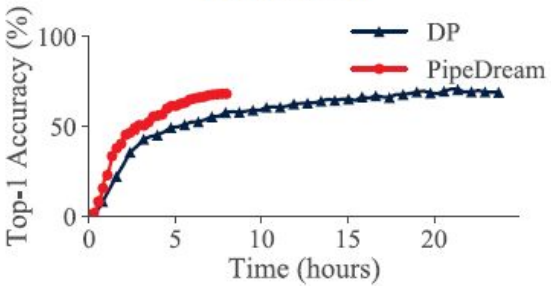
NYU SAI LAB

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

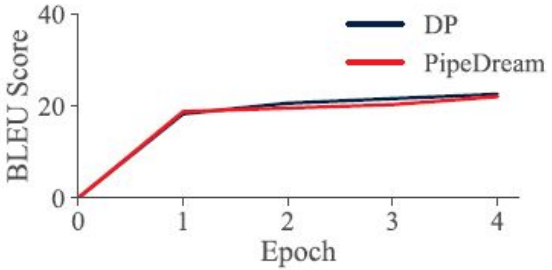
Model Parallelism: PipeDream



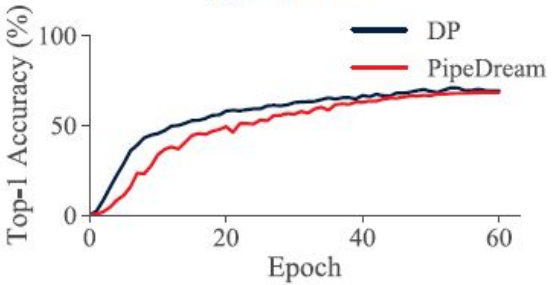
(a) Cluster-A.



(b) Cluster-B.



(a) GNMT-16

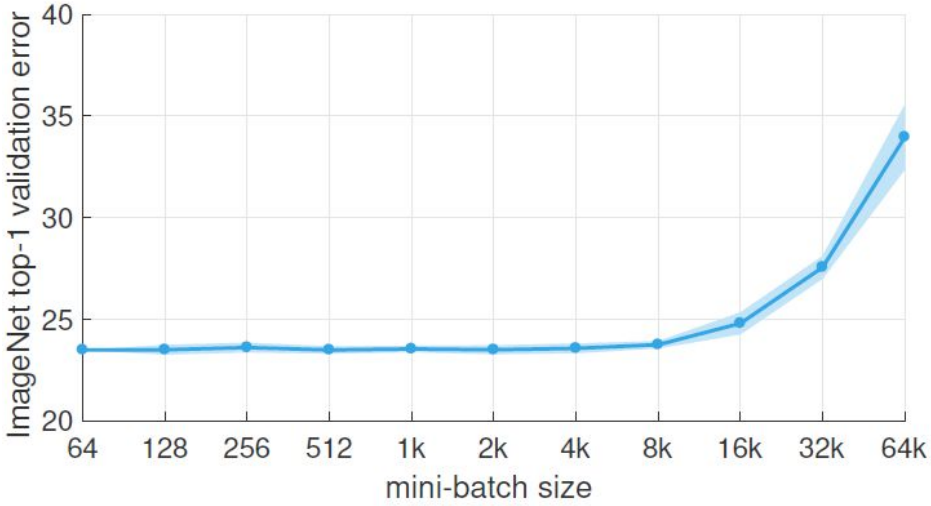


(b) VGG-16

NYU SAI LAB

Narayanan, Deepak, et al. "PipeDream: Generalized pipeline parallelism for DNN training." *Proceedings of the 27th ACM symposium on operating systems principles*. 2019.

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour



- For all minibatch sizes we set the learning rate as a linear function of the minibatch size and apply a simple warmup phase for the first few epochs of training.
- Using this simple approach, accuracy of our models is invariant to minibatch size (up to an 8k minibatch size).
- This enables a linear reduction in training time with 90% efficiency as we scale to large minibatch sizes, allowing us to train an accurate 8k minibatch ResNet-50 model in 1 hour on 256 GPUs.

Goyal, P. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

- Our goal is to use large minibatches in place of small minibatches while maintaining training and generalization accuracy.
  - CNNs typically use batch sizes ranging from 64 to 1K.
  - BERT models employ 8K–32K global batches with the LAMB optimizer and layerwise adaptive scaling.
  - Large language models (LLMs) are trained in large-scale distributed settings with batch sizes of 8K–64K tokens per step.
- $$L(w) = \frac{1}{|X|} \sum_{x \in X} l(x, w) \quad w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t)$$

Goyal, P. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

- **Linear Scaling Rule**: When the minibatch size is multiplied by k, multiply the learning rate by k.

$$w_{t+k} = w_t - \eta \frac{1}{n} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_{t+j}) \qquad \hat{w}_{t+1} = w_t - \hat{\eta} \frac{1}{kn} \sum_{j<k} \sum_{x \in \mathcal{B}_j} \nabla l(x, w_t)$$

- As the batch size increases, the number of iterations reduces, we need to increase the learning rate. If $\nabla l(x, w_t) \approx \nabla l(x, w_{t+j})$, then the two equations are equivalent.

Goyal, P. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).

Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour

- At the start of training, model parameters are usually randomly initialized and uncalibrated. A high learning rate at this point can cause large, erratic gradient updates that push parameters into unstable regions, leading to exploding losses or divergence.

- We start from a learning rate of η and increment it by a constant amount at each iteration such that it reaches $k\eta$ after 5 epochs. After the warm up, we go back to the original learning rate schedule.

Goyal, P. "Accurate, large minibatch SGD: training imagenet in 1 hour." *arXiv preprint arXiv:1706.02677* (2017).

Topics

- Speculative Decoding (Continue)
- Distributed DNN Training
- Distributed DNN Inference
- Federated Learning

BranchyNet

- Data samples are not equal in their recognition difficulties.
- For the easy samples, they only needs to be processed with a few layers before generating the correct results.

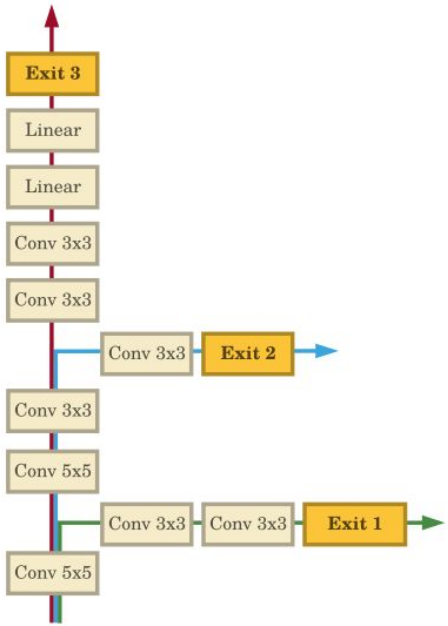
NYU SAI LAB

Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Branchynet: Fast inference via early exiting from deep neural networks." *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016.

BranchyNet

- During Inference, a confidence score is computed at each exit point, if greater than a predefined threshold, then the output is computed locally, leading to a faster inference.

- The confidence score is defined as: $\text{entropy}(\boldsymbol{y}) = \sum_{c \in \mathcal{C}} y_c \log y_c,$



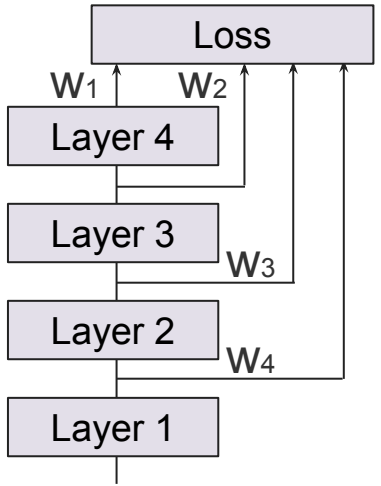
NYU SAI LAB

Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Branchynet: Fast inference via early exiting from deep neural networks." *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016.

BranchyNet

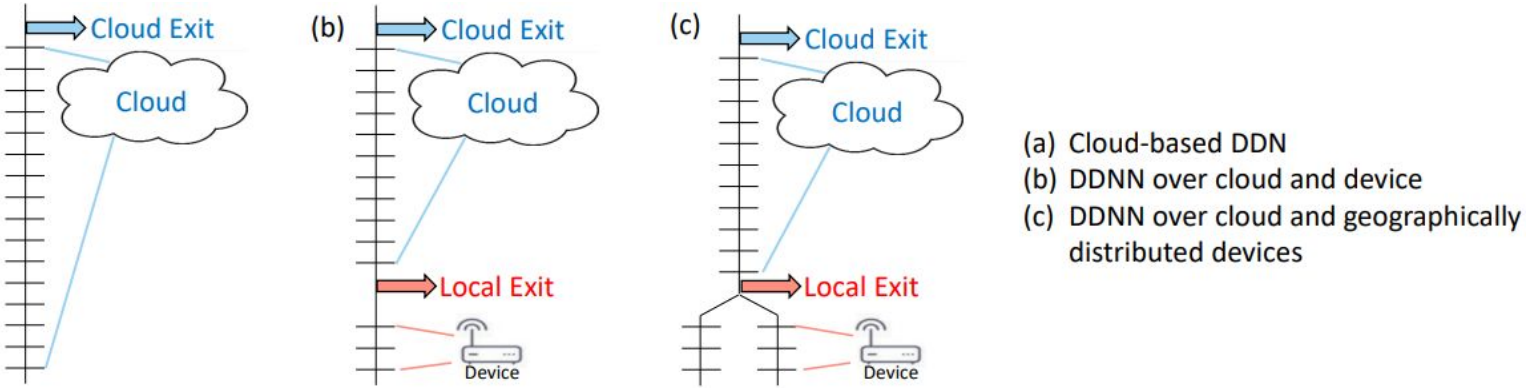
- To train the Branchy-style DNN, we can sum the cross-entropy loss at each local exit points, and train them jointly.

$$L_{\text{branchynet}}(\hat{\boldsymbol{y}}, \boldsymbol{y}; \theta) = \sum_{n=1}^{N} w_n L(\hat{\boldsymbol{y}}_{\text{exit}_n}, \boldsymbol{y}; \theta)$$



Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Branchynet: Fast inference via early exiting from deep neural networks." *2016 23rd international conference on pattern recognition (ICPR)*. IEEE, 2016.

Distributed Deep Neural Networks over the Cloud, the Edge and End Devices

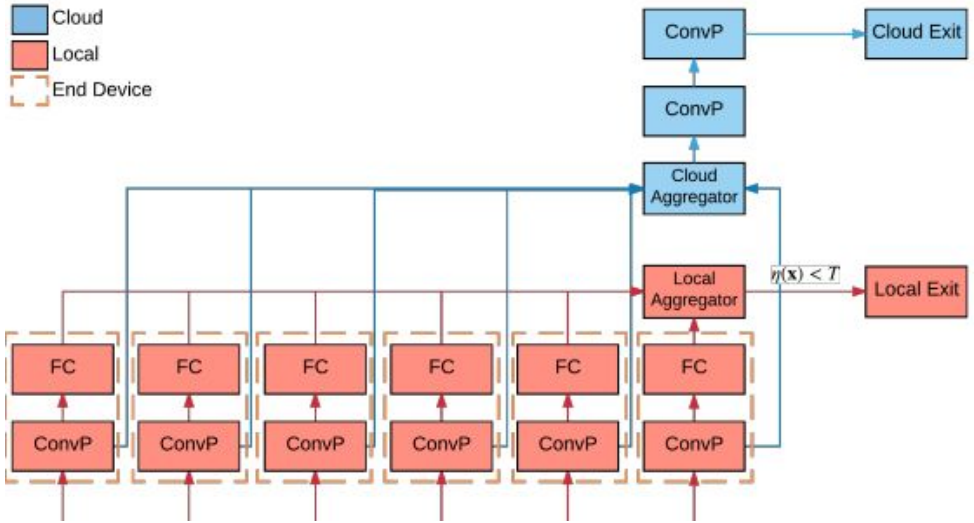


- We propose distributed deep neural networks (DDNNs) over distributed computing hierarchies, consisting of the cloud, the edge (fog) and end devices.

NYU SAI LAB

Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Distributed deep neural networks over the cloud, the edge and end devices." *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017.
Kang, Yiping, et al. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge." *ACM SIGARCH Computer Architecture News* 45.1 (2017): 615-629.

DDNN

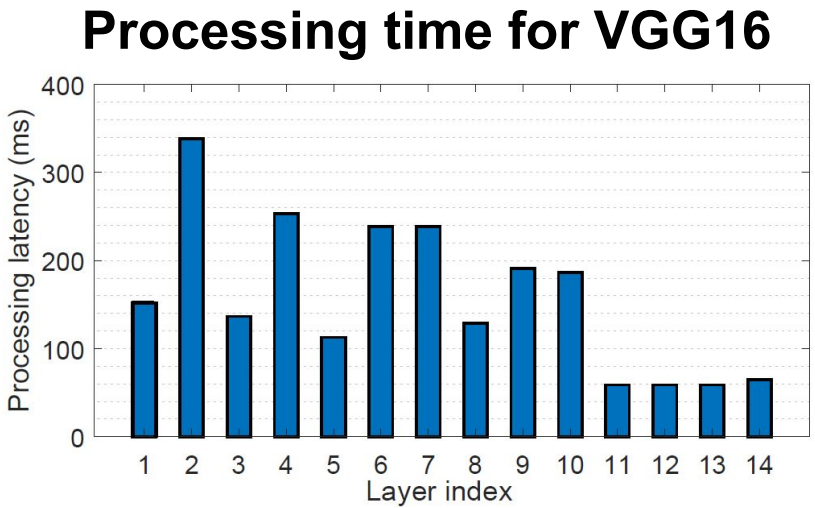


- Each edge device is implemented with a local DNN for local inference.
- The results from each local DNN is first aggregated locally.
- If the local exit is not confident, the activation output after the last convolutional layer from each end device is sent to the cloud aggregator for further processing.

NYU SAI LAB

Teerapittayanon, Surat, Bradley McDanel, and Hsiang-Tsung Kung. "Distributed deep neural networks over the cloud, the edge and end devices." *2017 IEEE 37th international conference on distributed computing systems (ICDCS)*. IEEE, 2017.
Kang, Yiping, et al. "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge." *ACM SIGARCH Computer Architecture News* 45.1 (2017): 615-629.

ADCNN

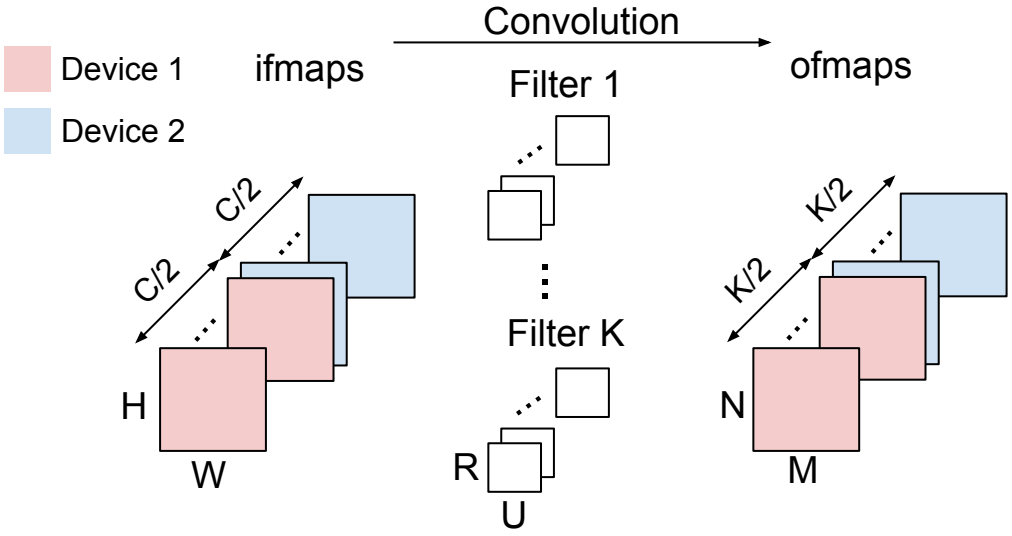


- Earlier layers take much longer to process than the later layers.

NYU SAI LAB

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "Adaptive distributed convolutional neural network inference at the network edge with ADCNN." *Proceedings of the 49th International Conference on Parallel Processing*. 2020.

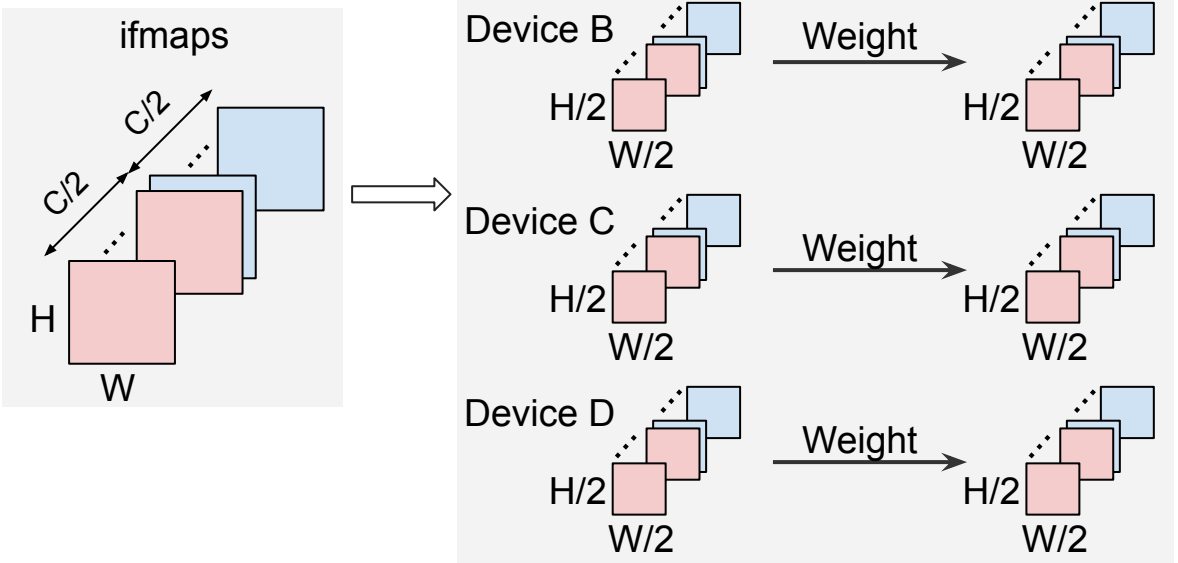
ADCNN



- In channelwise partition, each node needs to exchange their partially accumulated output feature maps to produce final output feature maps, which leads to a significant communication overhead.

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "Adaptive distributed convolutional neural network inference at the network edge with ADCNN." *Proceedings of the 49th International Conference on Parallel Processing*. 2020.

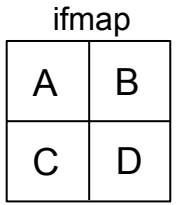
ADCNN



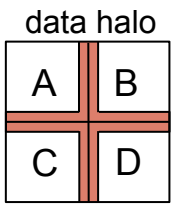
- The input will partitioned in spatial dimension and distribute over multiple devices.
- The weight will duplicate and save on each device.

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "Adaptive distributed convolutional neural network inference at the network edge with ADCNN." *Proceedings of the 49th International Conference on Parallel Processing*. 2020.

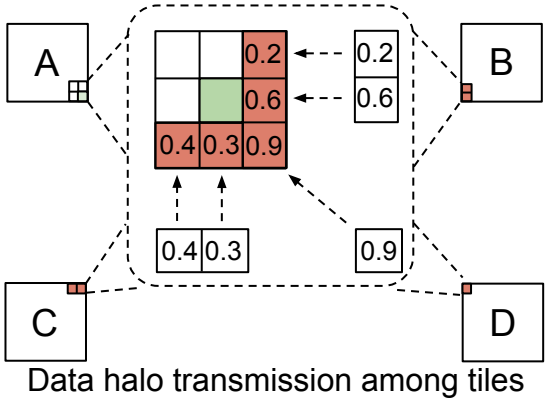
ADCNN



(a)



(b)

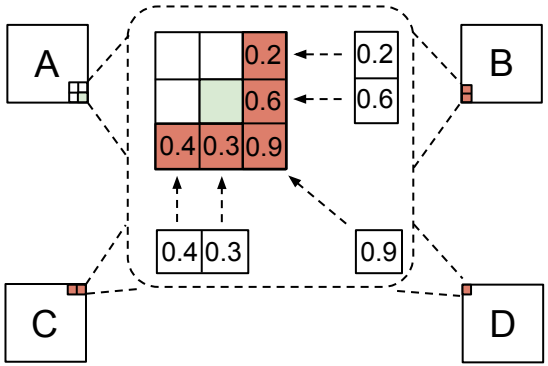


Data halo transmission among tiles

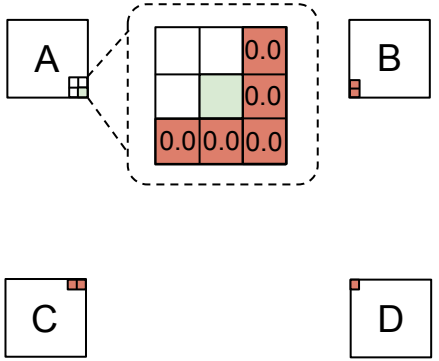
(c)

- In spatial partition, each tile needs to transmit their data halo in order to compute the correct result.

ADCNN



Normal Spatial Partition



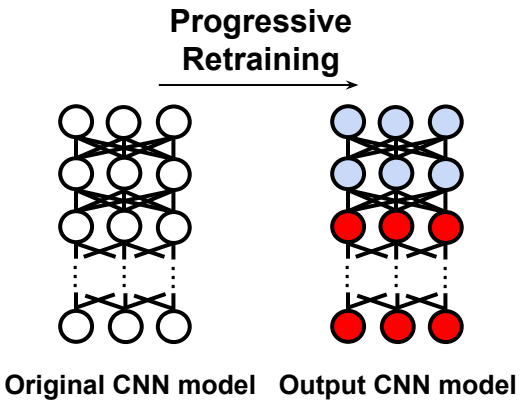
Fully Decomposable Spatial Partition (FDSP)

- The cross-tile information transfer can be eliminated by padding the edge pixels with zeros.

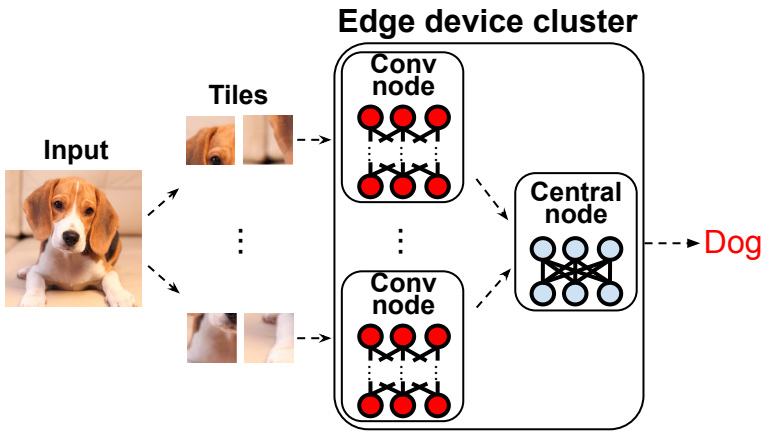
NYU SAI LAB

ADCNN

Step 1

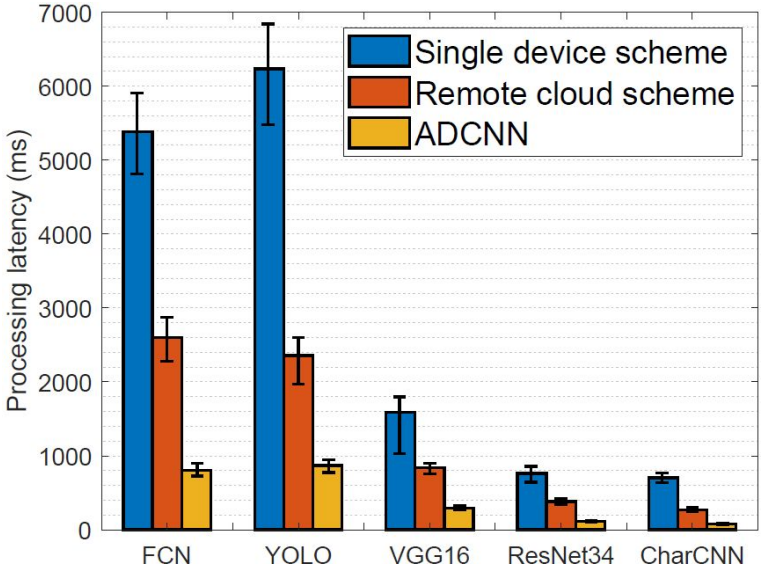


Step 2



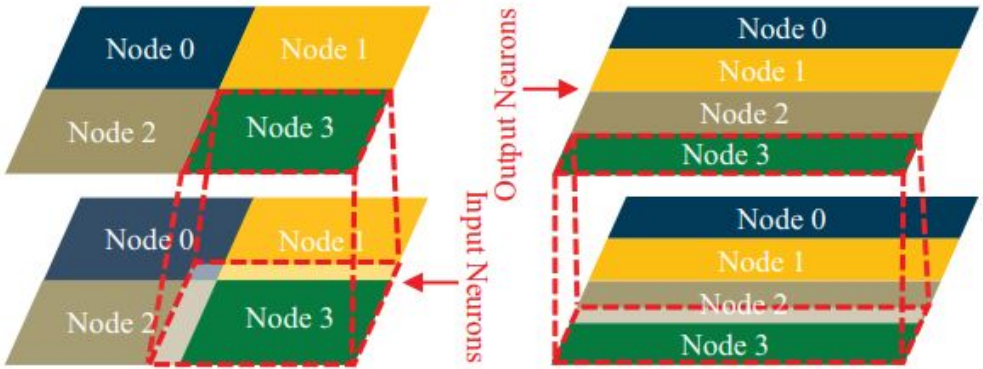
Evaluation Results

- We implement ADCNN system with nine identical Raspberry Pi devices which simulate the edge devices. Among these nine devices, eight are used as Conv nodes, and the rest one is used as the Central node.
- Baselines:
  - Single device scheme
  - Remote cloud scheme
- ADCNN decreases the average processing latency by 6.68x and 4.42x, respectively.



NYU SAI LAB

MoDNN



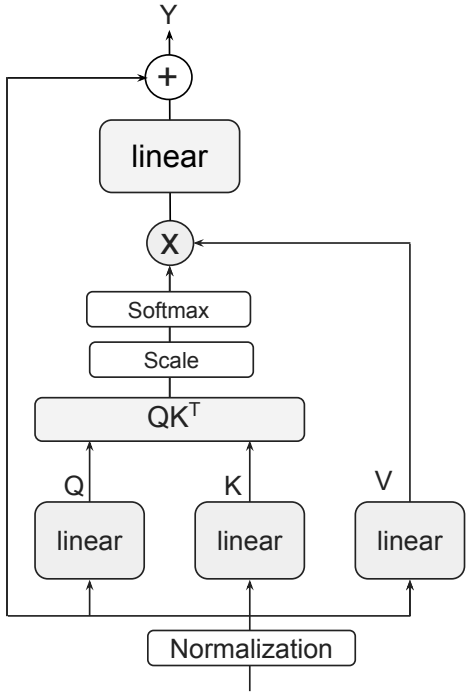
(a) Neurons in 2D-grids of 4 nodes. (b) BODP for 4 worker nodes.

NYU SAI LAB

Mao, Jiachen, et al. "Modnn: Local distributed mobile computing system for deep neural network." *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE, 2017.

Distributed Inference for Transformer

- Input dimension: L ✖ E
- Partitioning across dimension L requires all-reduce operation for QK^T computation.
- Partitioning across dimension E (headwise partition) is better:
  - Q, K, V tensors are broken into multiple components along the embedding dimension.
    - (B,L,E) ✖ (E ✖ E) → (B ✖ L ✖ E)
    - (B,L,E) → (B, M, L, E/M) → (B, M , L, D) , where D=E/M
  - All the following operations can be performed independently over each head M.
    - QK^T →(B, M, L ✖ D) ✖ (B, M, D ✖ L) → (B, M, L ✖ L)
    - Softmax(QK^T) → (B, M, L ✖ L)
    - Softmax(QK^T) ✖ V → (B, M, L ✖ L) ✖ (B, M, L ✖ D) → (B, M, L ✖ D) → (B ✖ L ✖ E)



NYU SAI LAB

Topics

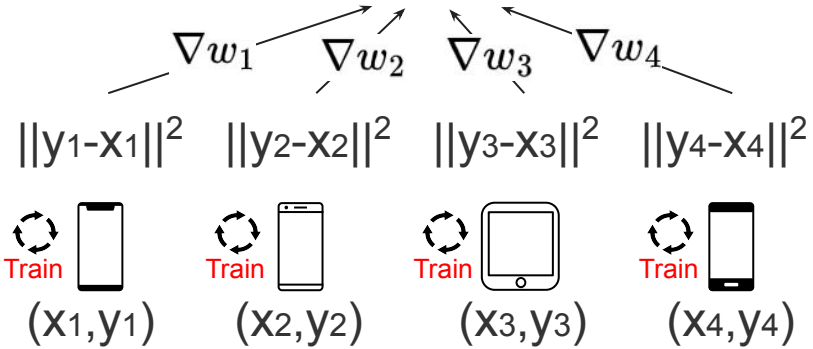
- Speculative Decoding (Continue)
- Distributed DNN Training
- Distributed DNN Inference
- Federated Learning

Federated Learning

- Training data: (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)

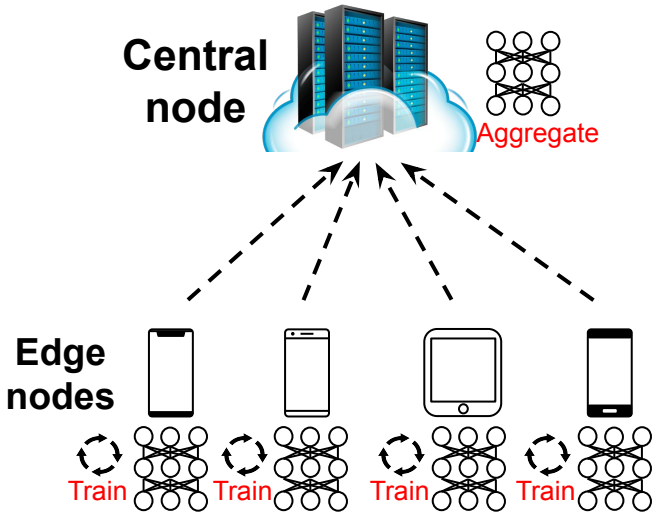
$$\sum_{i=1}^{4} ||y_i - F(x_i)||^2$$

$$\nabla w = \frac{\nabla w_1 + \nabla w_2 + \nabla w_3 + \nabla w_4}{4}$$



- Non-iid training data distribution
- Heterogeneity among the edge devices
- Communication error

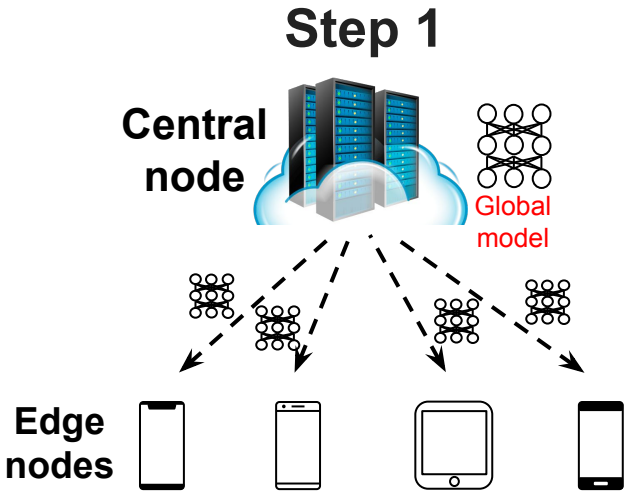
Federated Learning



- Federated learning is a machine learning technique that allows the training of models across multiple decentralized nodes holding local data samples, **without exchanging their data**.
- This approach enhances privacy, user can train the powerful DNN model **without sharing the dataset**.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

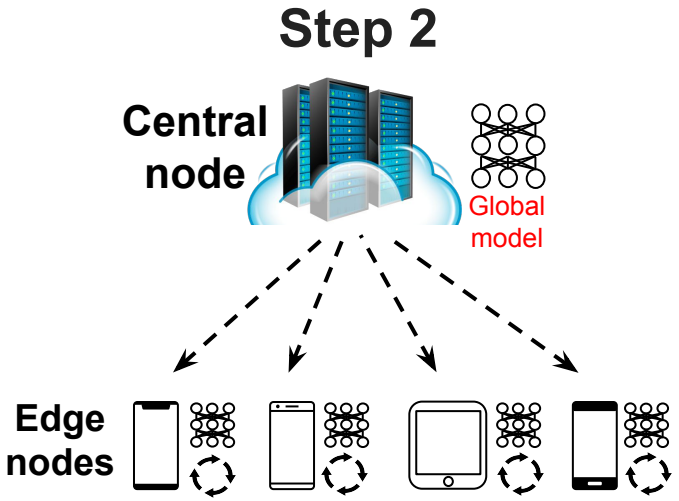


- A global model is initialized on the central node and sent to all participating nodes .

$$w_i = w_{global} \quad \text{For each i}$$

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

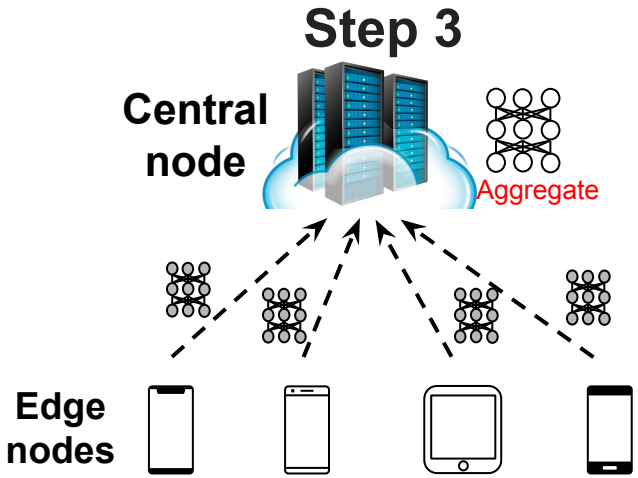


$$w_i' = \underset{w_i}{min} L(F(D_i), Y_i)$$

- Each node i trains the global model locally using its own data for a few epochs.
- The length of local training process may vary.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg

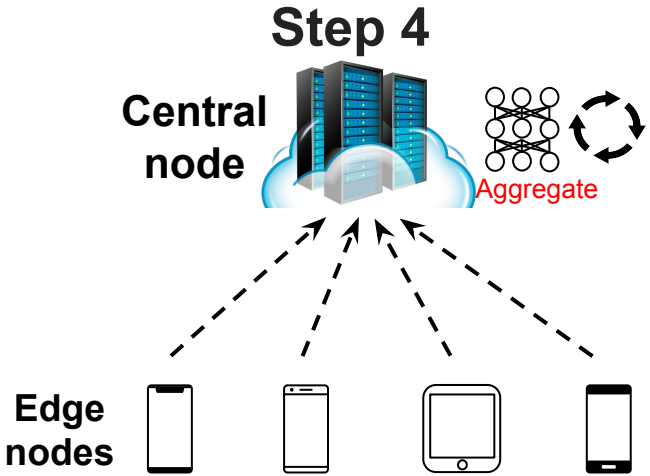


$$\Delta w_i = w - w_i$$

- Local updates are sent from each node to the central node.

McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

FedAvg



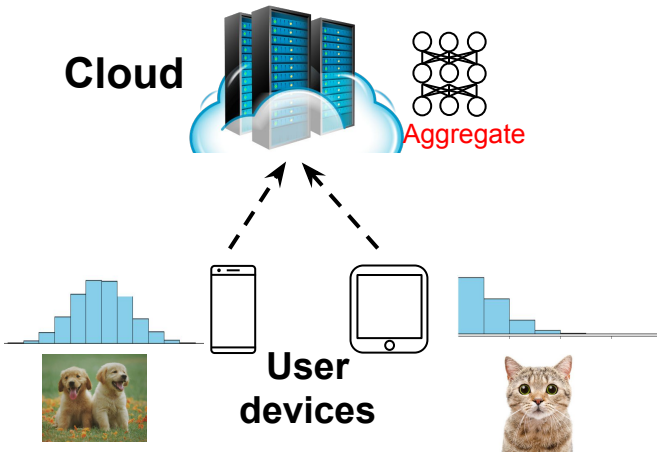
- The central node aggregates the local updates to update the global model.

$$w + \frac{1}{N} \sum_i \Delta w_i$$

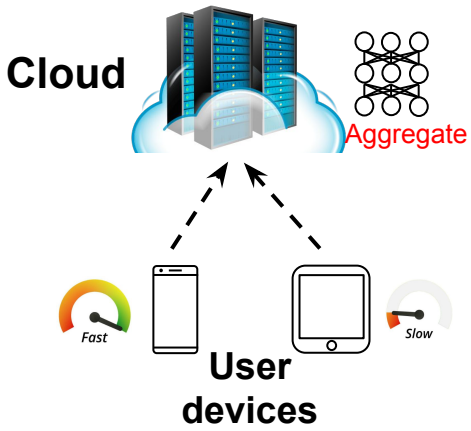
McMahan, Brendan, et al. "Communication-efficient learning of deep networks from decentralized data." *Artificial intelligence and statistics*. PMLR, 2017.

Federated Learning Problems: Non-IID

- However, in FL, the data distributed across different devices or clients is not drawn from the same statistical distribution.
- Unlike the scenario distributed training, where the training data are randomly distributed. For FL, the data stored in each device is highly biased.
- This may lead to significant accuracy degradation for the global model.

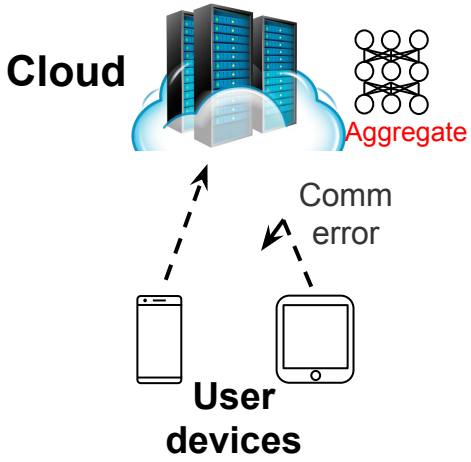


Federated Learning Problems: Heterogeneity



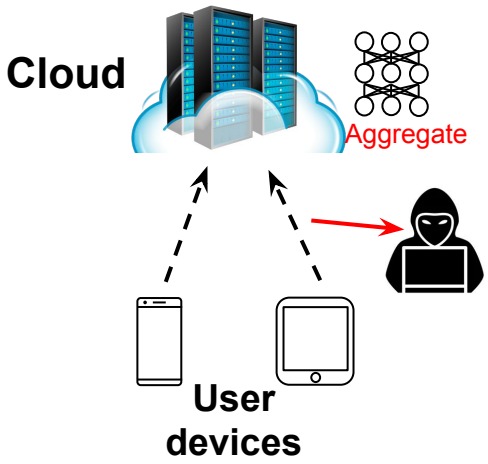
- Different edge device may have different processing speed.
- This will cause the total latency of each training round bottlenecked by the straggler, leading to a slow convergence of the training process.

Federated Learning Problems: Communication



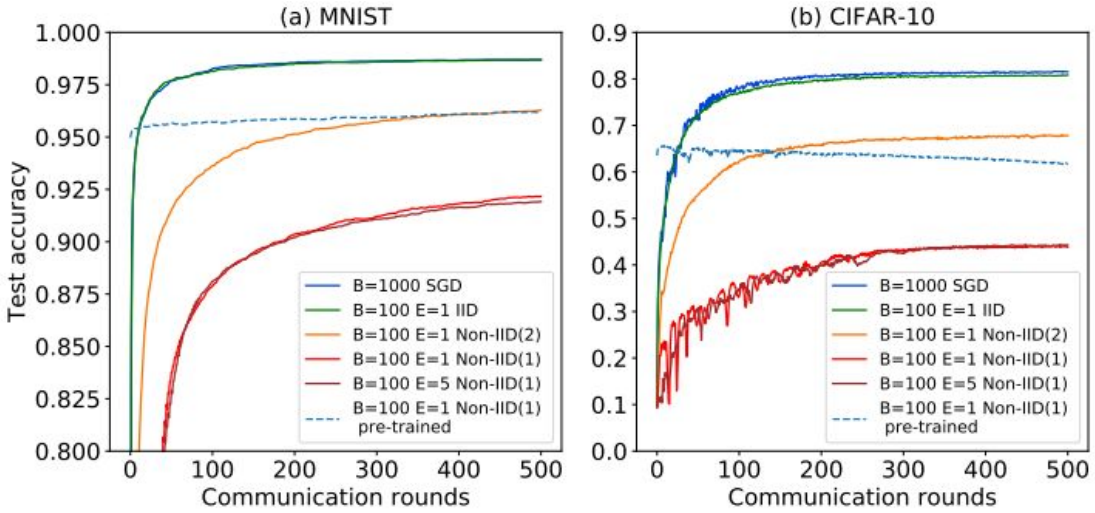
- The communication between edge devices and central cloud may incur transmission loss or error.
- This will impact the training latency and accuracy.

Federated Learning Problems: Privacy



- The attacker can leverage the transmitted gradient to reconstruct the original input training data.
- This will lead to privacy leakage.

Federated Learning with Non-iid Data

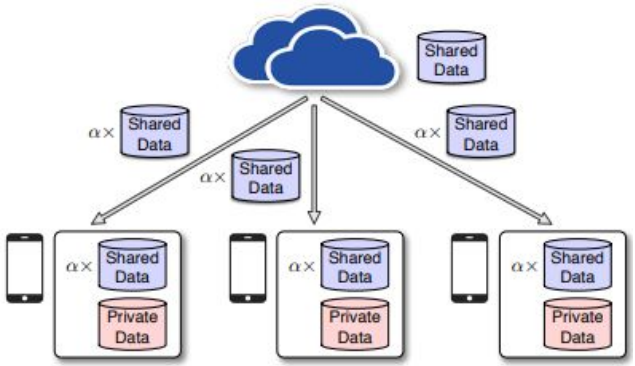


- The training sets are evenly partitioned into 10 clients.
- For IID setting, each client is randomly assigned a uniform distribution over 10 classes.
- For non-IID setting, the data is sorted by class and divided to create two extreme cases: (a) 1-class non-IID, where each client receives data partition from only a single class, and (b) 2-class non-IID, where the sorted data is divided into 20 partitions and each client is randomly assigned 2 partitions from 2 classes.

Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

Federated Learning with Non-iid Data

- We propose a data-sharing strategy to improve FedAvg with non-IID data by creating a small subset of data which is globally shared between all the edge devices.
- Experiments show that test accuracy can be increased by ~30% on CIFAR-10 dataset with only 5% globally shared data.



Zhao, Yue, et al. "Federated learning with non-iid data." *arXiv preprint arXiv:1806.00582* (2018).

FedProx

Algorithm 2 `FedProx` (Proposed Framework)

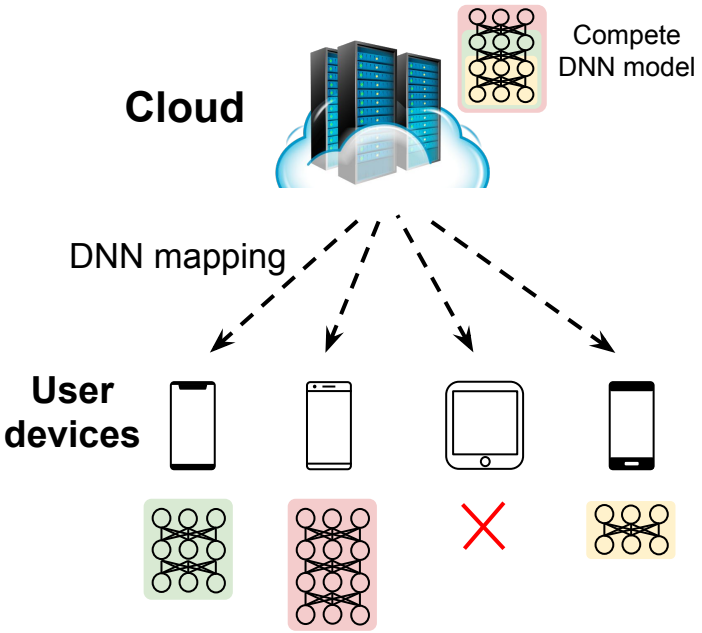
Input: $K, T, \mu, \gamma, w^0, N, p_k, k = 1, \cdots, N$
for $t = 0, \cdots, T-1$ **do**
  Server selects a subset S_t of K devices at random (each device k is chosen with probability p_k)
  Server sends w^t to all chosen devices
  Each chosen device $k \in S_t$ finds a w_k^{t+1} which is a γ_k^t-inexact minimizer of: $w_k^{t+1} \approx \arg\min_w \; h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$
  Each device $k \in S_t$ sends w_k^{t+1} back to the server
  Server aggregates the w's as $w^{t+1} = \frac{1}{K}\sum_{k \in S_t} w_k^{t+1}$
end for

$$\min_w h_k(w;\; w^t) = F_k(w) + \frac{\mu}{2}\|w - w^t\|^2$$

- We add an extra term to minimize the l_2 distance between the initial weight w_t and the learned weight w.
- This loss ensures that the learnt w is not too different from the original w.

Li, Tian, et al. "Federated optimization in heterogeneous networks." *Proceedings of Machine learning and systems* 2 (2020): 429-450.

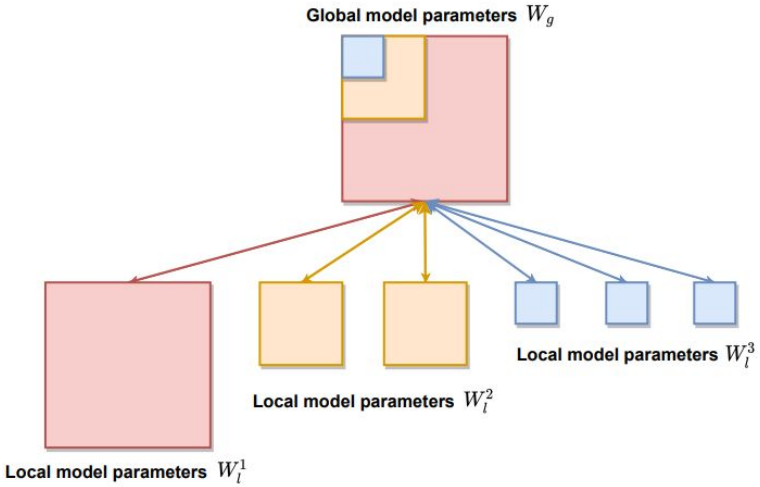
Federated Learning Problems: Heterogeneity



- End devices will have heterogeneous system configuration.
- HeteroFL partitions and assigns the DNN based on the processing power of each device.
- Each device only train a subset of the DNN model.

Diao, Enmao, Jie Ding, and Vahid Tarokh. "Heterofl: Computation and communication efficient federated learning for heterogeneous clients." *arXiv preprint arXiv:2010.01264* (2020).

HeteroFL



- Each edge device will be assigned with part of the neural network to perform local training based on its computational complexity.

Diao, Enmao, Jie Ding, and Vahid Tarokh. "Heterofl: Computation and communication efficient federated learning for heterogeneous clients." *arXiv preprint arXiv:2010.01264* (2020).

Federated Learning Problems: Communication

$$e(\mathbf{u}, \bar{\mathbf{u}}) = \frac{1}{N} \sum_{j=1}^{N} I(\mathrm{sgn}(u_j) = \mathrm{sgn}(\bar{u}_j))$$

- u_j denotes the sign of the model weight after local updates.
- Our solution dynamically identifies relevant local updates and excludes those irrelevant from being.
- Only the local device with high relevance will transmit their weight to the central server.

Luping, W.; Wei, W.; and Bo, L. 2019. Cmfl: Mitigating communication overhead for federated learning. In 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), 954–964. IEEE.

FedMARL

Final model Accuray

Total Training Latency

Total Bandwidth

$$\max_{A} E\left[w_1 Acc(T) - w_2 \sum_{t \in T} H_t - w_3 \sum_{t \in T} B_t\right]$$

$$A = [a_n^t]$$

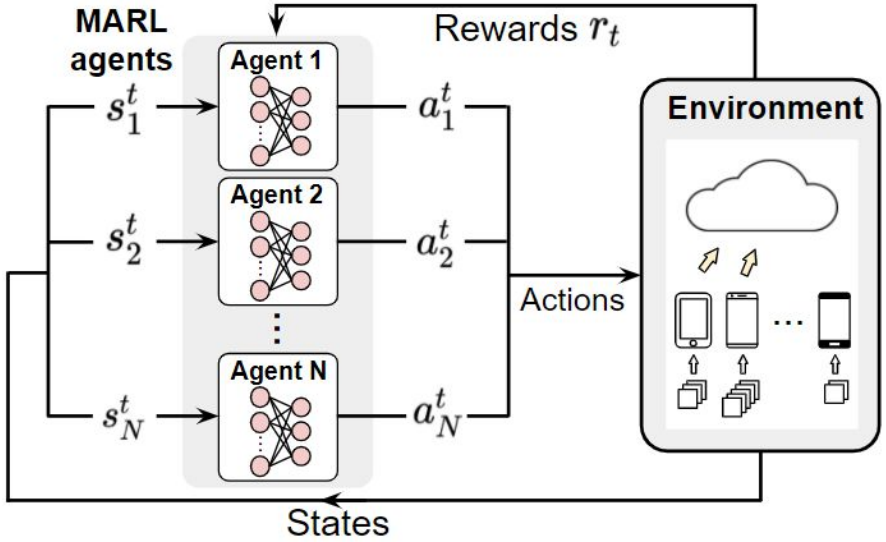
Client Selection

- Our objective is to maximize the accuracy of the global model while minimizing the total processing latency and communication cost.
- w1,w2,w3 are the importance of the objectives controlled by the FL application designers.
- The FL optimization problem is difficult to solve directly. We instead model the problem as a MARL problem.

NYU SAI LAB

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "A multi-agent reinforcement learning approach for efficient client selection in federated learning." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. No. 8. 2022.

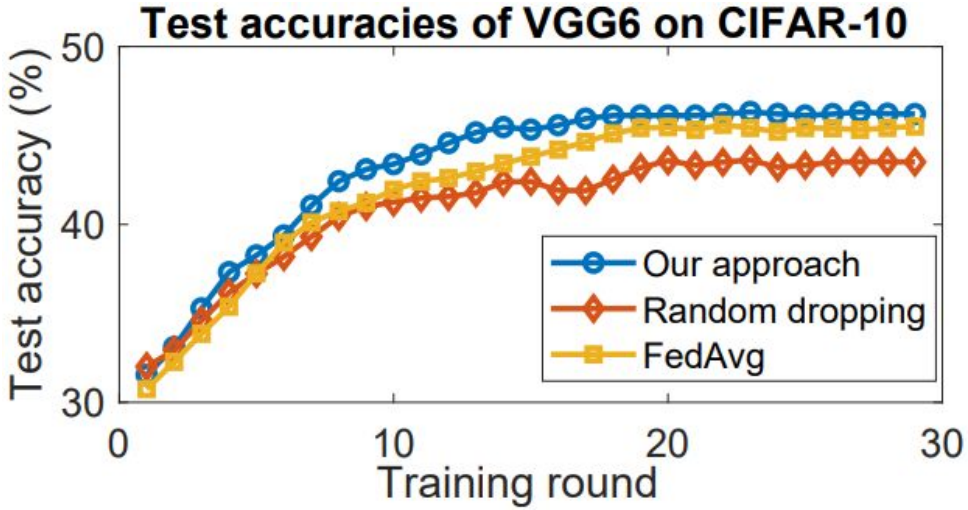
FedMARL



- In FedMarl, each client device n relies on an MARL agent at the central server to make its participation decision. Each MARL agent contains a simple two-layer Multi-layer perceptron (MLP) that is cheap to implement.

Zhang, Sai Qian, Jieyu Lin, and Qi Zhang. "A multi-agent reinforcement learning approach for efficient client selection in federated learning." *Proceedings of the AAAI conference on artificial intelligence*. Vol. 36. No. 8. 2022.

FedMARL



- Every random dropping is better than FedAvg.
- FedMarl is much better than random dropping and FedAvg.