Memory Footprint Reduction Techniques for DNN Training: An Overview

Episode III. Weight Placement for EX-Large Models

Bojian Zheng

2022/2/18

Outline

- EX-Large Models: Bring us better accuracy but also challenges.
- Existing Solutions: Data and Model Parallelism
- ZeRO-Infinity: Data-Parallelism-based Solution for EX-Large Training

Recall: A History of Prior Works

2015-2016 | **2016-** | **2019-**

Weight Pruning for Efficient **Inference**

Feature Maps Reduction for Efficient **Training**

Weight Placement for **EX-Large Models**

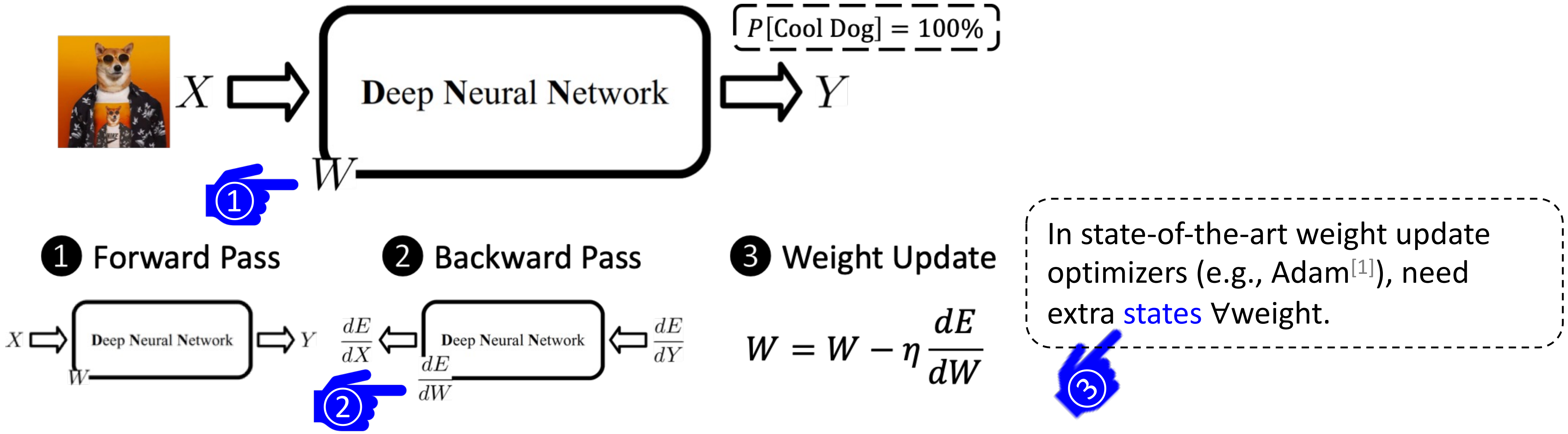
Topic for Today!

[1] T. Chen et al. *DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning*. ASPLOS 2014
[2] S. Han et al. *EIE: Efficient Inference Engine on Compressed Deep Neural Network*. ISCA 2015
[3] S. Han et al. *Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding*. ICLR 2015
[4] Y. Chen et al. *Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks*. ISCA 2016
…

[5] T. Chen et al. *Training Deep Nets with Sublinear Memory Cost*. arXiv 2016
[6] M. Rhu et al. *vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design*. MICRO 2016
[7] A. Jain et al. *Gist: Efficient Data Encoding for Deep Neural Network Training*. ISCA 2018
[8] B. Zheng et al. *Echo: Compiler-based GPU Memory Footprint Reduction for LSTM RNN Training*. ISCA 2020
…

[9] S. Rajbhandari et al. *ZeRO-Infinity: Breaking the GPU Memory Wall for Extreme Scale Deep Learning*. SC 2021
…

Recall: GPU Memory Allocations in DNN Training



- Major GPU memory consumers: **Model States** & Feature Maps
  - **Model States**: weights (①), gradients (②), optimizer states (③)

[1] D. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. ICLR 2015

Why EX-Large Models?

- Rule of Thumb in Deep Learning:

  Larger & Deeper Models ⇒ **Better Model Accuracy**[1, 2]

- *The amount of computation resources consumed by a job can reject the* ***problem scale*** *and may also indicate the* ***commercial value*** *of the workload*[3].

[1] K. He et al. *Deep Residual Learning for Image Recognition*. CVPR 2016
[2] T. Brown et al. *Language Models are Few-Shot Learners*. NeurIPS 2020
[3] M. Wang et al. *Characterizing Deep Learning Training Workloads on Alibaba-PAI*. IISWC 2019

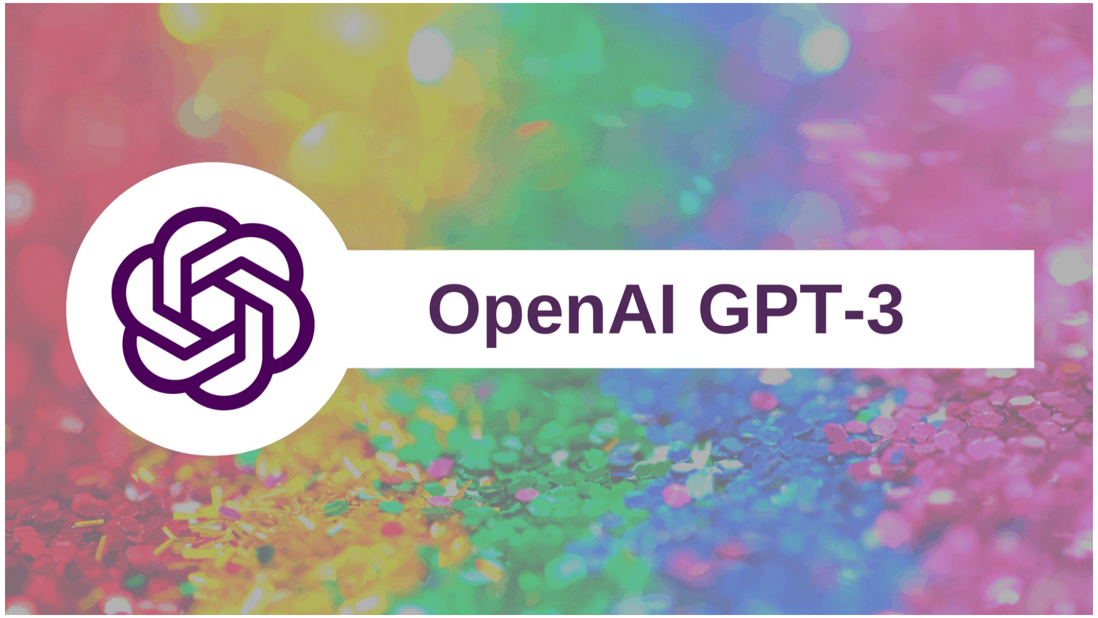
Challenges with Training EX-Large Models

- BERT[1]
  - State-of-the-Art Natural Language Processing Model (2018)
  - 340M parameters
  - Cannot fit into the GPU memory even with a batch size of 1[2].
  - **Addressable** with offloading[3], checkpointing[4] etc.

[1] J. Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. NAACL 2019
[2] https://github.com/google-research/bert
[3] M. Rhu et al. *vDNN: Virtualized Deep Neural Networks for Scalable, Memory-Efficient Neural Network Design*. MICRO 2016
[4] T. Chen et al. *Training Deep Nets with Sublinear Memory Cost*. arXiv 2016

Challenges with Training EX-Large Models

- GPT-3[1]
  - State-of-the-Art Natural Language Processing Model (**2020**)
  - 175**B** parameters ($470\times$ LARGER)
  - 700 GB storage $\gg$ modern GPU memory capacity
  - 3×700 GB = 2.1 TB more for gradients & optimizer states



[1] T. Brown et al. *Language Models are Few-Shot Learners*. NeurIPS 2020

Challenges with Training EX-Large Models

- Multi-Interests[1, 2, 3]
  - Industry-scale recommender
  - Encodes every object of interest into a vector, which assembles into an **embedding table**.
  - Up to 240 GB storage ≫ modern GPU memory capacity
  - Cannot fit a single layer into the GPU memory!

[1] P. Covington et al. *Deep Neural Networks for YouTube Recommendations*. RecSys 2016

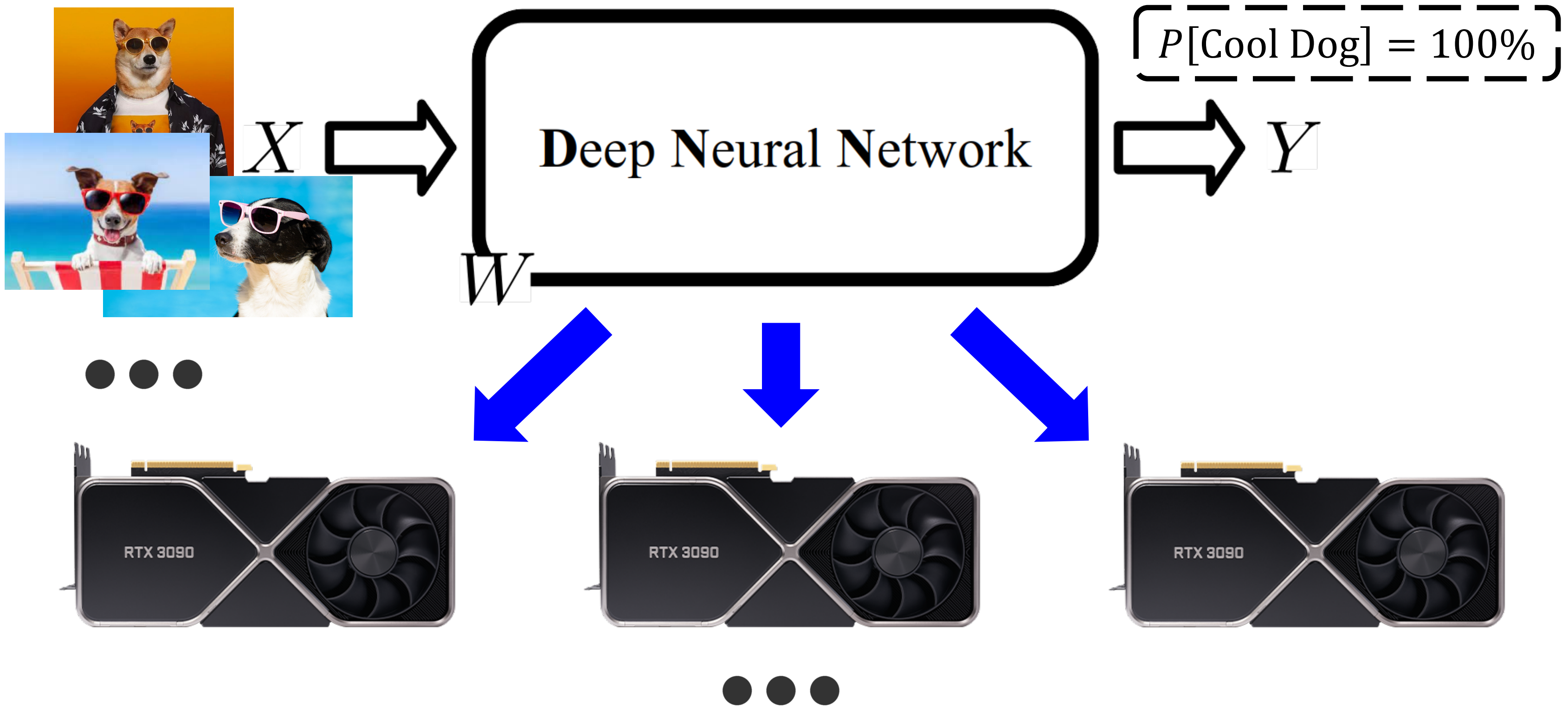
[2] J. Weston et al. *Nonlinear Latent Factorization by Embedding Multiple User Interests*. RecSys 2013

[3] M. Wang et al. *Characterizing Deep Learning Training Workloads on Alibaba-PAI*. IISWC 2019

Today's DNN Training In A Nutshell

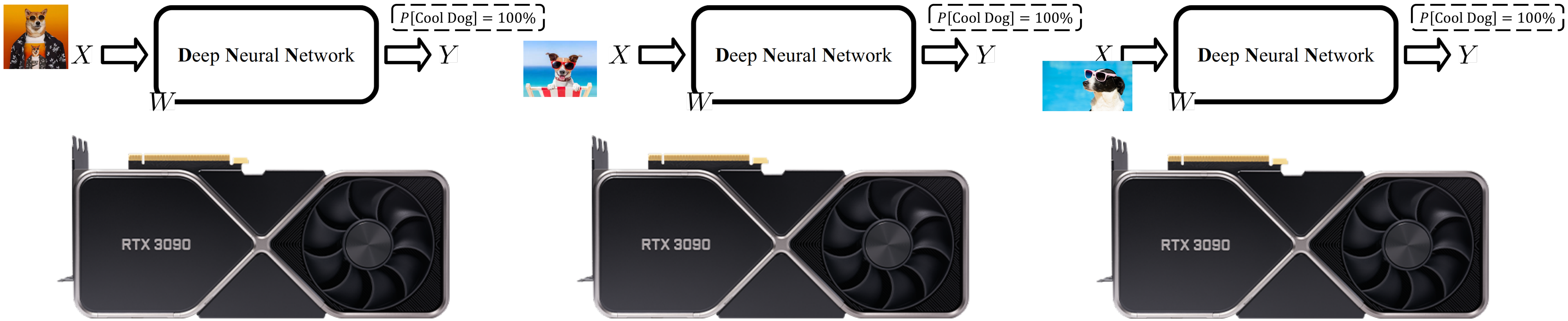
Data and Model Parallelism

- Partition into multiple devices? E.g., Data and/or Model Parallelism?



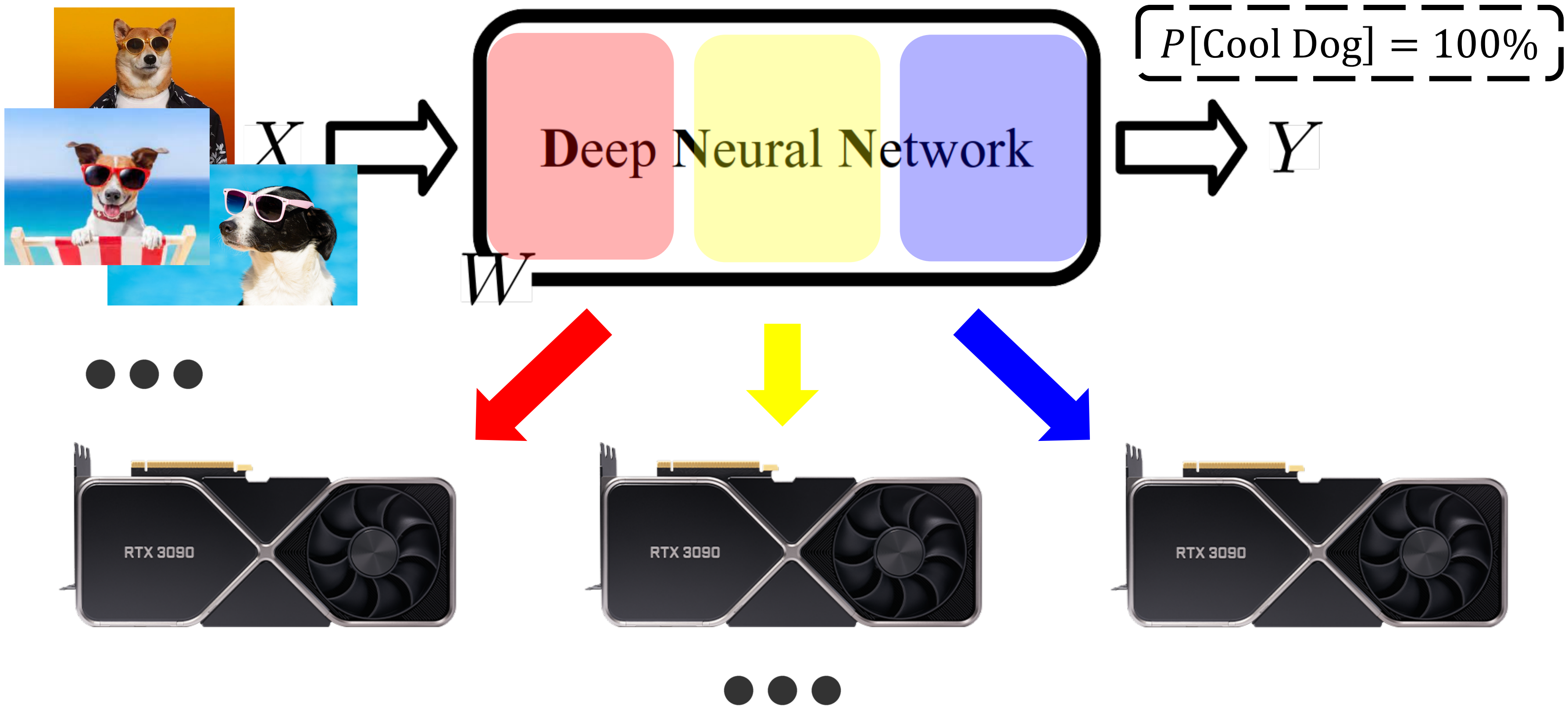
Data Parallelism

- Partitions the **data** into multiple devices.



Model Parallelism

- Partitions the **model** into multiple devices.



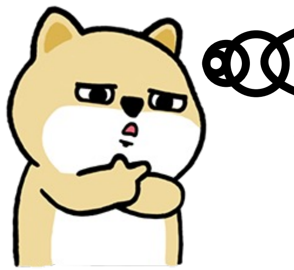
Data vs. Model Parallelism

Data Parallelism

+
+
– Need to replicate the entire model. Challenges of the EX-large models persist!

Model Parallelism

– Need to significantly refactor the code for load-balancing.
– Hard to support models with complex dependencies.
+



Could we have the merits from both?
i.e., generic support for EX-Large models with minor code changes?

ZeRO[1]

- A solution that is based on data-parallelism.
- Key Idea Partitions model states among devices and use efficient communication primitives to gather them.
- However, hungry DNN models need much more memory!

[1] S. Rajbhandari et al. *ZeRO: Memory Optimizations toward Training Trillion Parameter Models*. SC 2020

ZeRO-Infinity[1]

- ZeRO[2] + Infinity Offload Engine
- Observation GPUs are also accompanied with a CPU and NVMe SSDs

	GPUs	CPU	NVMe SSDs
Memory (TB)	0.5	1.5	28

- Key Idea Partitions model states among (GPUs + CPU + SSD) and Keeps most of the states to the latter two.

[1] S. Rajbhandari et al. *ZeRO: Memory Optimizations toward Training Trillion Parameter Models*. SC 2020

Challenges

- Challenge #1 How to efficiently get the weights from CPU/SSDs?

- Challenge #2 How to handle cases when the weight of a single layer cannot fit into the GPU memory?

Key Idea #1. Weight Gathering

- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker *broadcast* the weight to others.

Nodes	GPUs	Aggregate GPU Memory (TBs)	Aggregate CPU Memory (TBs)	Aggregate NVME Memory (TBs)	GPU Memory Bandwidth/GPU (GB/s)	GPU-GPU Bandwidth (GB/s)	CPU-GPU Memory Bandwidth/GPU (GB/s)	NVME-GPU Memory Bandwidth/GPU (GB/s)
1	1	0.032	1.5	28.0	600-900	N/A	12.0	12.0
1	16	0.5	1.5	28.0	600-900	150-200	4.1	1.6
4	64	2.0	6.0	112.0	600-900	60-100	4.1	1.6
16	256	8.0	24.0	448.0	600-900	60-100	4.1	1.6
64	1024	32.0	96.0	1792.0	600-900	60-100	4.1	1.6

Key Idea #1. Weight Gathering

- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker *broadcast* the weight to others.

Nodes	GPUs	Aggregate GPU Memory (TBs)	Aggregate CPU Memory (TBs)	Aggregate NVME Memory (TBs)	GPU Memory Bandwidth/GPU (GB/s)	GPU-GPU Bandwidth (GB/s)	CPU-GPU Memory Bandwidth/GPU (GB/s)	NVME-GPU Memory Bandwidth/GPU (GB/s)
1	1	0.032	1.5	28.0	600-900	N/A	12.0	12.0
1	16	0.5	1.5	28.0	600-900	150-200	4.1	1.6
4	64	2.0	6.0	112.0	600-900	60-100	4.1	1.6
16	256	8.0	24.0	448.0	600-900	60-100	4.1	1.6
64	1024	32.0	96.0	1792.0	600-900	60-100	4.1	1.6

Key Idea #1. Weight Gathering

- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker *broadcast* the weight to others.

Nodes	GPUs	Aggregate GPU Memory (TBs)	Aggregate CPU Memory (TBs)	Aggregate NVME Memory (TBs)	GPU Memory Bandwidth/GPU (GB/s)	GPU-GPU Bandwidth (GB/s)	CPU-GPU Memory Bandwidth/GPU (GB/s)	NVME-GPU Memory Bandwidth/GPU (GB/s)
1	1	0.032	1.5	28.0	600-900	N/A	12.0	12.0
1	16	0.5	1.5	28.0	600-900	150-200	4.1	1.6
4	64	2.0	6.0	112.0	600-900	60-100	4.1	1.6
16	256	8.0	24.0	448.0	600-900	60-100	4.1	1.6
64	1024	32.0	96.0	1792.0	600-900	60-100	4.1	1.6

Key Idea #1. Weight Gathering

- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker *broadcast* the weight to others.

Nodes	GPUs	Aggregate GPU Memory (TBs)	Aggregate CPU Memory (TBs)	Aggregate NVME Memory (TBs)	GPU Memory Bandwidth/GPU (GB/s)	GPU-GPU Bandwidth (GB/s)	CPU-GPU Memory Bandwidth/GPU (GB/s)	NVME-GPU Memory Bandwidth/GPU (GB/s)
1	1	0.032	1.5	28.0	600-900	N/A	12.0	12.0
1	16	0.5	1.5	28.0	600-900	150-200	4.1	1.6
4	64	2.0	6.0	112.0	600-900	60-100	4.1	1.6
16	256	8.0	24.0	448.0	600-900	60-100	4.1	1.6
64	1024	32.0	96.0	1792.0	600-900	60-100	4.1	1.6

Same effective bandwidth even if 1024 GPUs are reading in parallel

Key Idea #1. Weight Gathering

- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
  - Better utilization than having a single worker *broadcast* the weight to others.

Nodes	GPUs	Aggregate GPU Memory (TBs)	Aggregate CPU Memory (TBs)	Aggregate NVME Memory (TBs)	GPU Memory Bandwidth/GPU (GB/s)	GPU-GPU Bandwidth (GB/s)	CPU-GPU Memory Bandwidth/GPU (GB/s)	NVME-GPU Memory Bandwidth/GPU (GB/s)
1	1	0.032	1.5	28.0	600-900	N/A	12.0	12.0
1	16	0.5	1.5	28.0	600-900	150-200	4.1	1.6
4	64	2.0	6.0	112.0	600-900	60-100	4.1	1.6
16	256	8.0	24.0	448.0	600-900	60-100	4.1	1.6
64	1024	32.0	96.0	1792.0	600-900	60-100	4.1	1.6

4/1.6 TB/s (Gather) vs. 12/12 GB/s (Broadcast)

Key Idea #1. Pipelined Prefetch

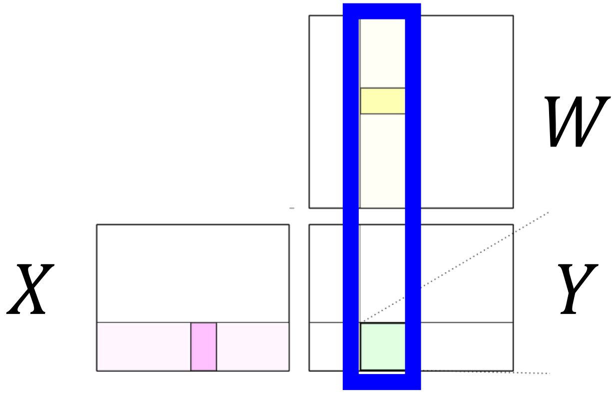
- Challenge #1 How to efficiently get the weights from CPU/SSDs?
- Optimization #1 ∀weight, each GPU worker holds a portion of it and **gathers** the rest from others.
- Optimization #2 **Pipelined prefetch** from SSD→CPU→GPU.

i	SSD	CPU	GPU	Compute
$i+1$		SSD	CPU	GPU
$i+2$			SSD	CPU
$i+3$				SSD

Read the weight slice from SSD for layer $i+3$ while layer i is computing its results.

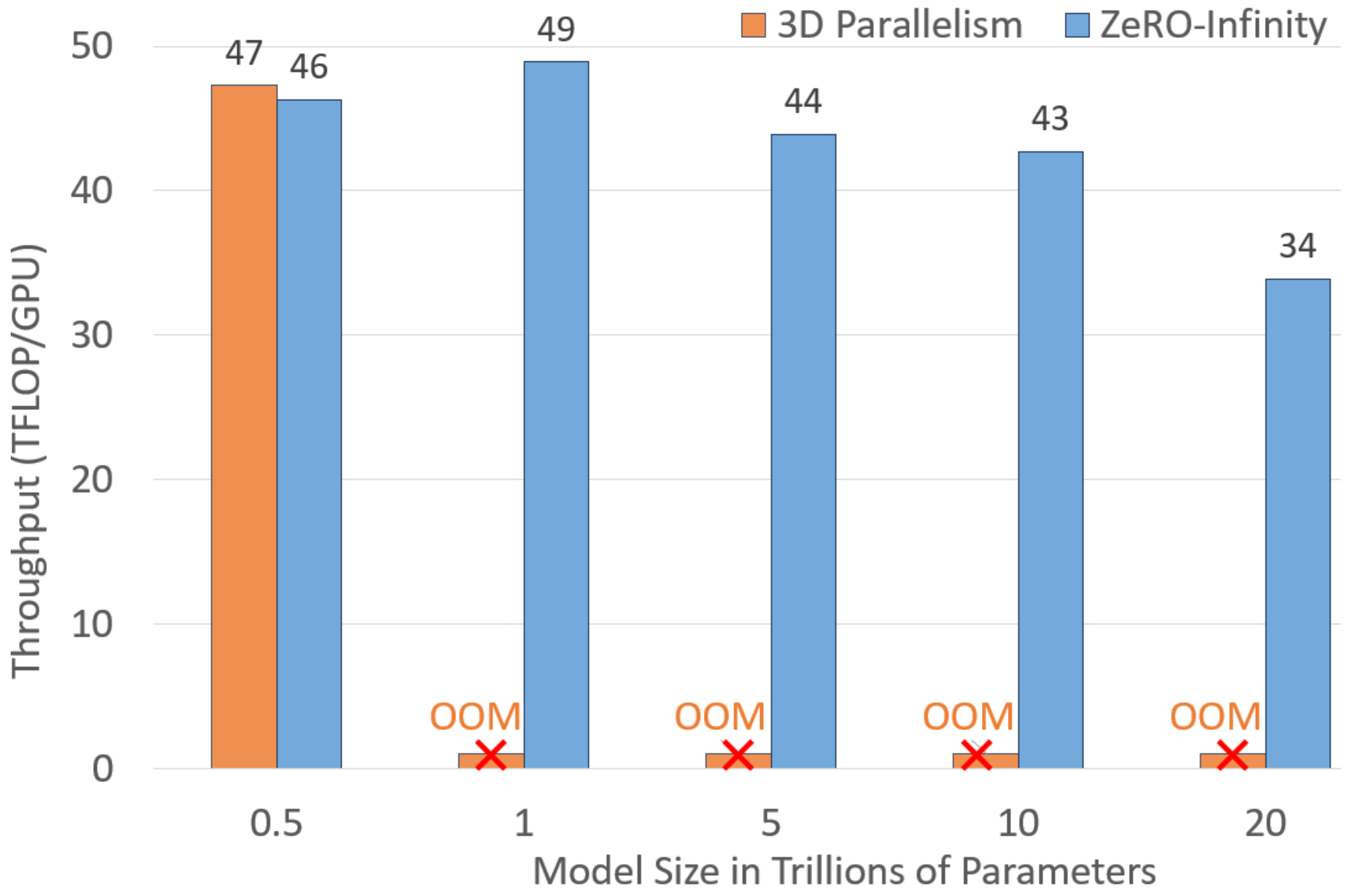
Key Idea #3. Tiled Computation

- Challenge #2 How to handle cases when the weight of a single layer cannot fit into the GPU memory?
- Key Idea Some layers can evaluate their results in **tiles** and some do not need the whole weight for the full results[1].
  - E.g., Dense Layer $Y = XW^T$



[1] M. Wang et al. *Characterizing Deep Learning Training Workloads on Alibaba-PAI*. IISWC 2019

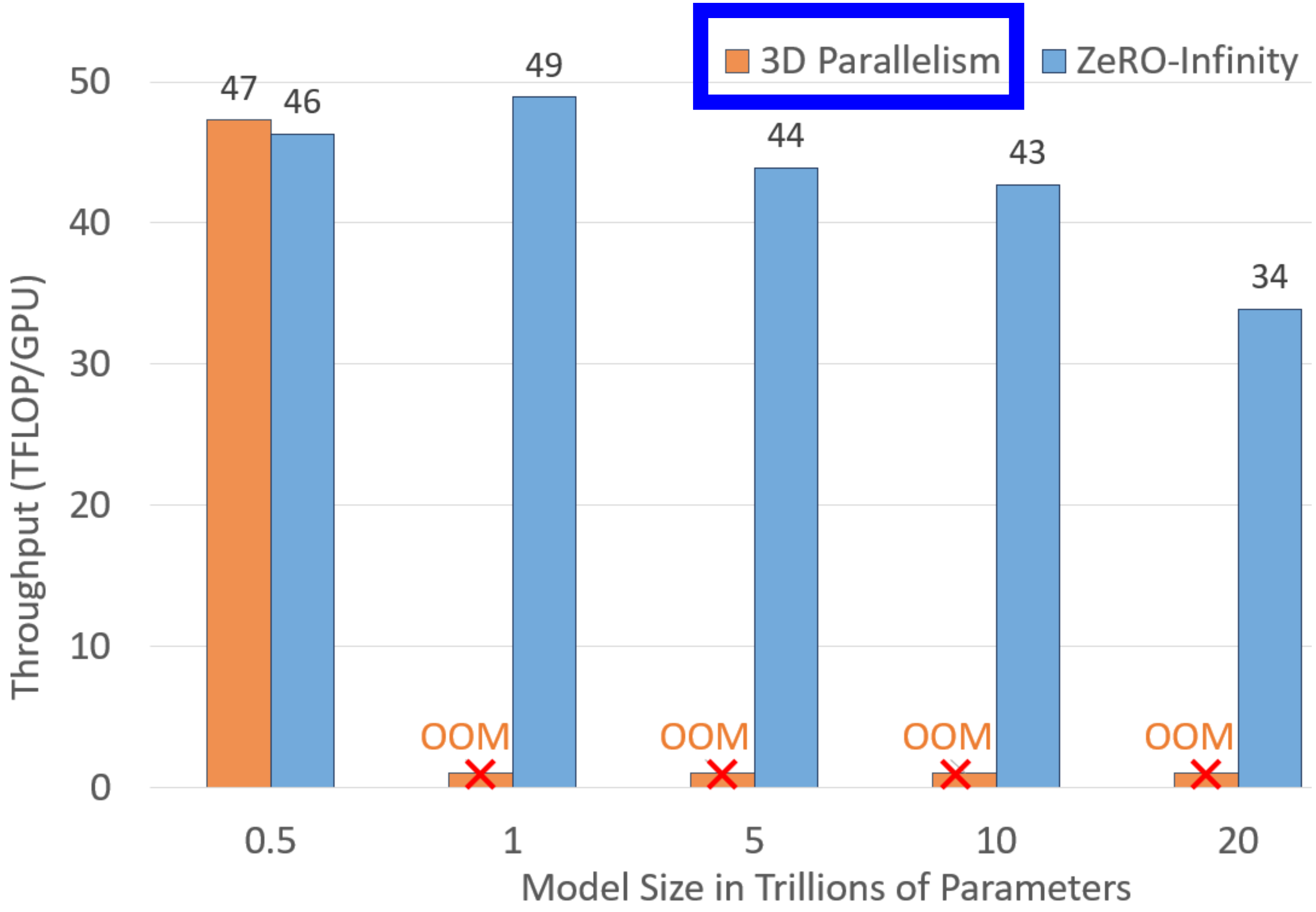
Key Results



3D Parallelism
ZeRO-Infinity
Throughput (TFLOP/GPU)
50
40
30
20
10
0
47
46
49
44
43
34
OOM
OOM
OOM
OOM
0.5
1
5
10
20
Model Size in Trillions of Parameters

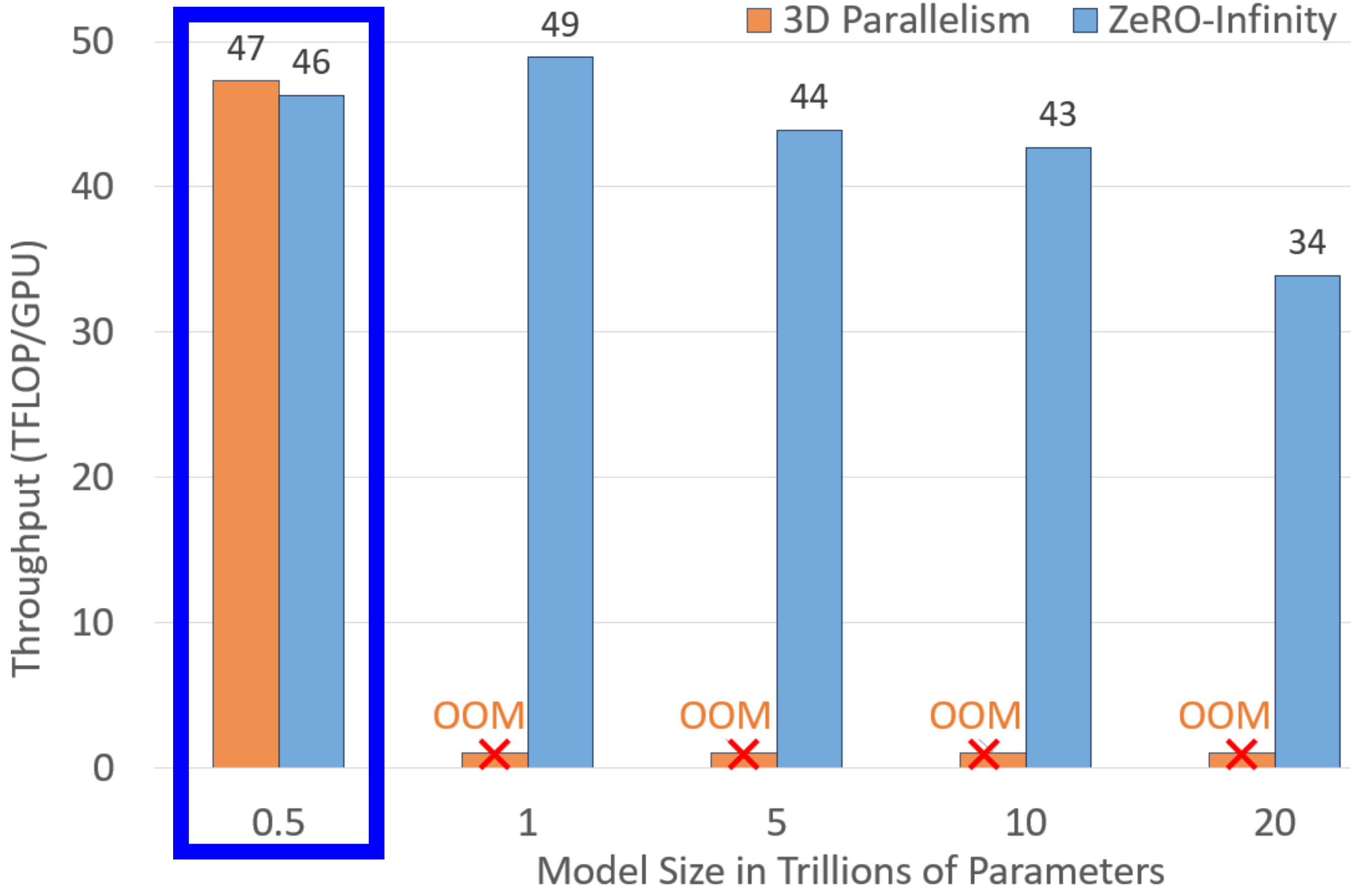
Key Results

Baseline with Data + Model Parallelism

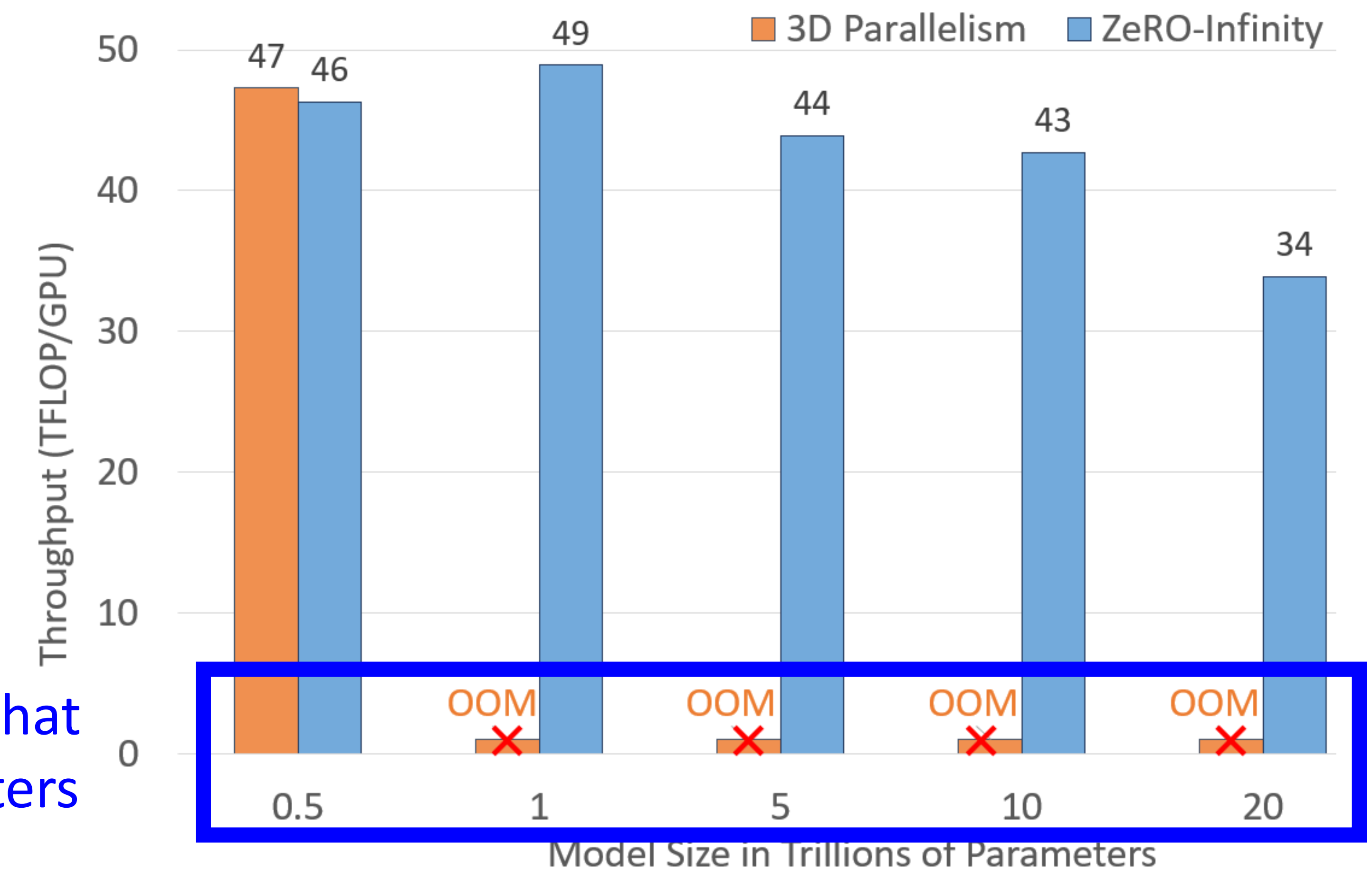


Key Results

Offloading to CPU/SSDs cause **2%** performance degradation



Key Results



Support for models that have trillion parameters

Conclusions

- EX-Large Models: Bring us better accuracy but also challenges.
- Existing Solutions: Data and Model Parallelism
  - Can’t make EX-large training practical while being easily programmable.
- ZeRO-Infinity: Data-Parallelism-based Solution for EX-Large Training
  - Key Ideas: Weight Gathering, Pipelined Prefetch, Tiled Computation
  - Makes it possible to train models with trillion parameters with small performance overhead (2%).
- https://github.com/microsoft/DeepSpeed

Memory Footprint Reduction Techniques for DNN Training: An Overview

Episode III. Weight Placement for EX-Large Models

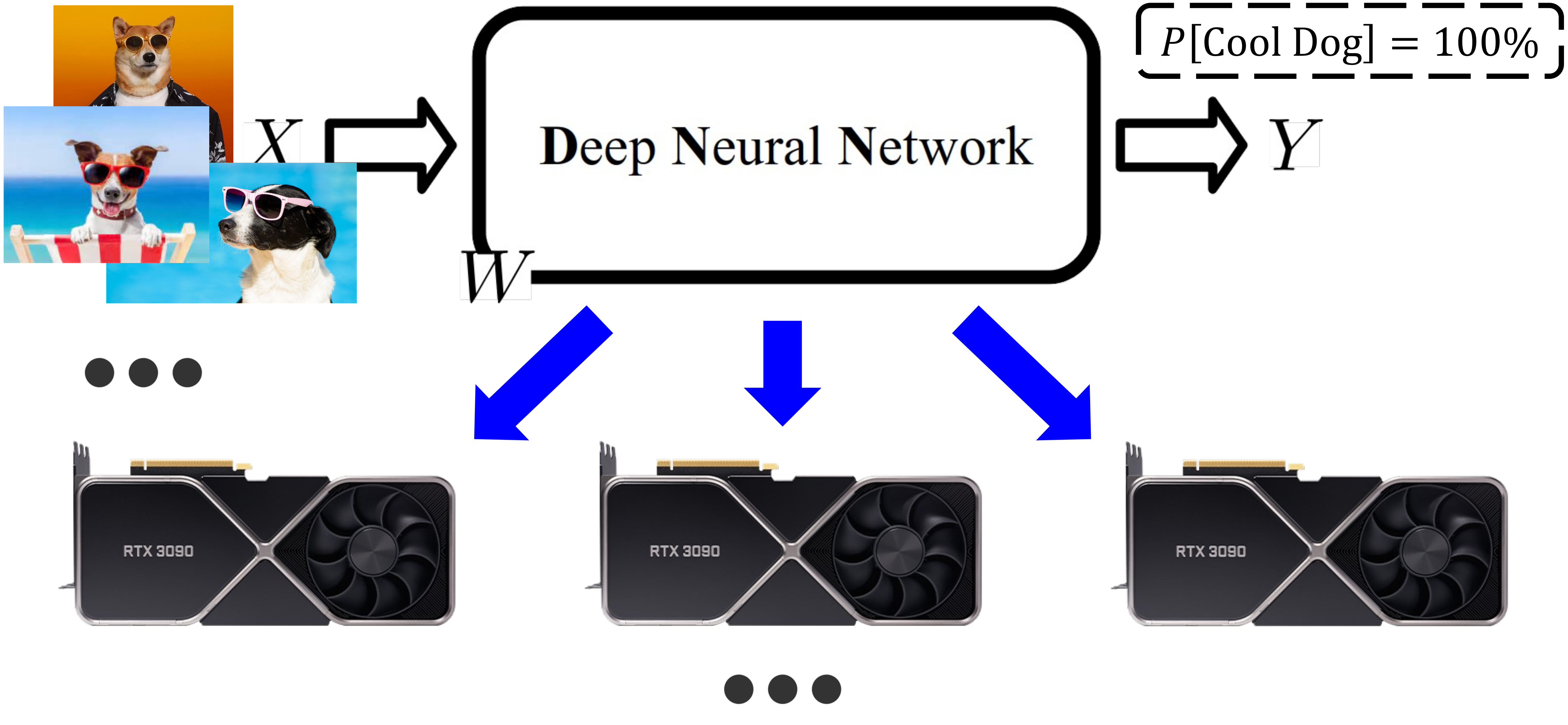
Bojian Zheng

2022/2/18

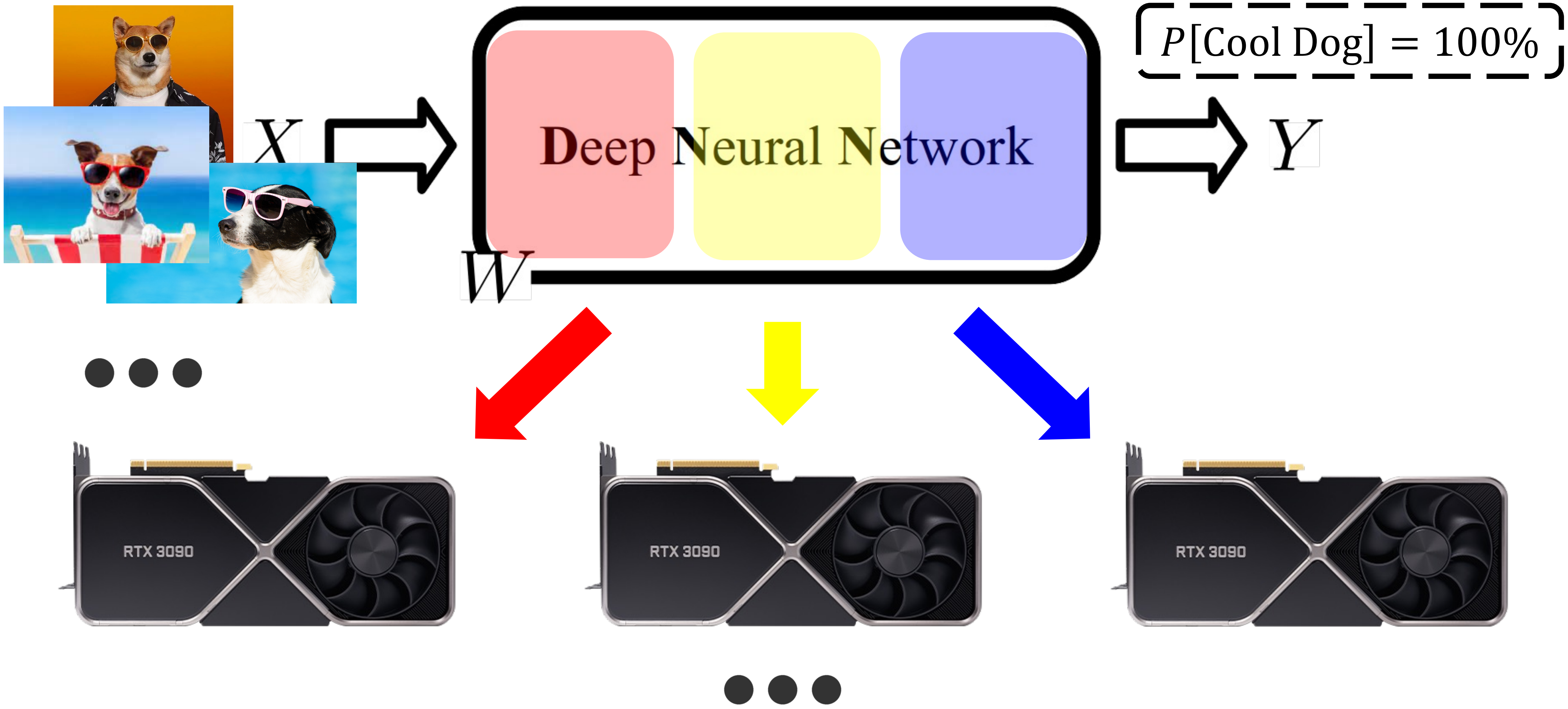
Why Not?

- Reduce the precision of the weight parameters[1]? E.g., FP32→FP16?
- **Challenge**: Weights need to be in high-precision format to accommodate for incremental gradient updates.

- E.g., Type equation here.



$P[\text{Cool Dog}] = 100\%$
X
Deep Neural Network
Y
W
RTX 3090
RTX 3090
RTX 3090

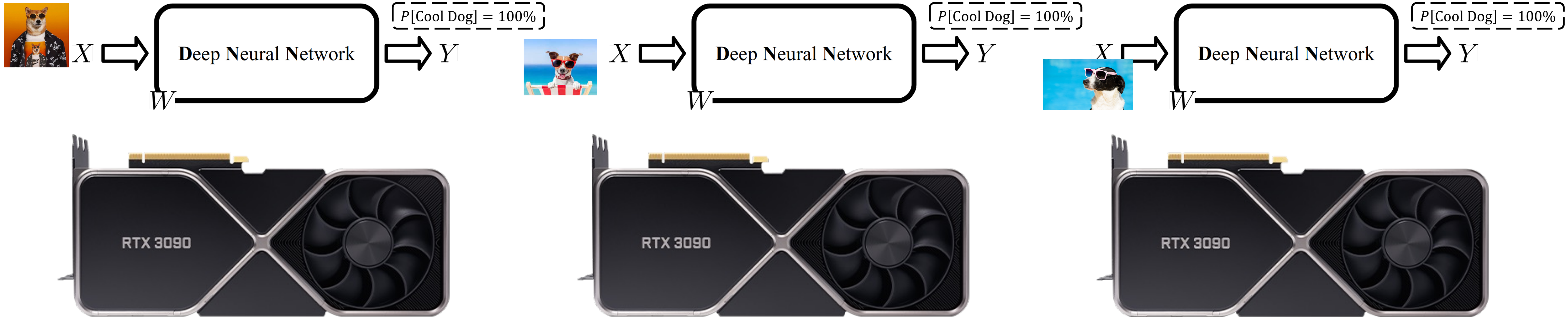


X
Deep Neural Network
W
$P[\text{Cool Dog}] = 100\%$
Y
RTX 3090
RTX 3090
RTX 3090

ZeRO-Infinity[1]

- ZeRO[2] + Infinity Offload Engine
- Observation GPUs are accompanied with large CPU memory and
- Key Idea Partitions model states among workers (GPUs + CPU + NVMe SSD). Offload the states in the latter two

[1] S. Rajbhandari et al. *ZeRO: Memory Optimizations toward Training Trillion Parameter Models*. SC 2020



X
Deep Neural Network
W
Y
P[Cool Dog] = 100%
RTX 3090
X
Deep Neural Network
W
Y
P[Cool Dog] = 100%
RTX 3090
X
Deep Neural Network
W
Y
P[Cool Dog] = 100%
RTX 3090