



Slides available at:

go.osu.edu/hoti25-hidl

Principles and Practice of Scalable and Distributed Deep Neural Networks Training and Inference

Tutorial at Hoti '25

by

Dhabaleswar K. (DK) Panda

The Ohio State University

panda@cse.ohio-state.edu

<http://www.cse.ohio-state.edu/~panda>

Nawras Alnaasan

The Ohio State University

alnaasan.1@osu.edu

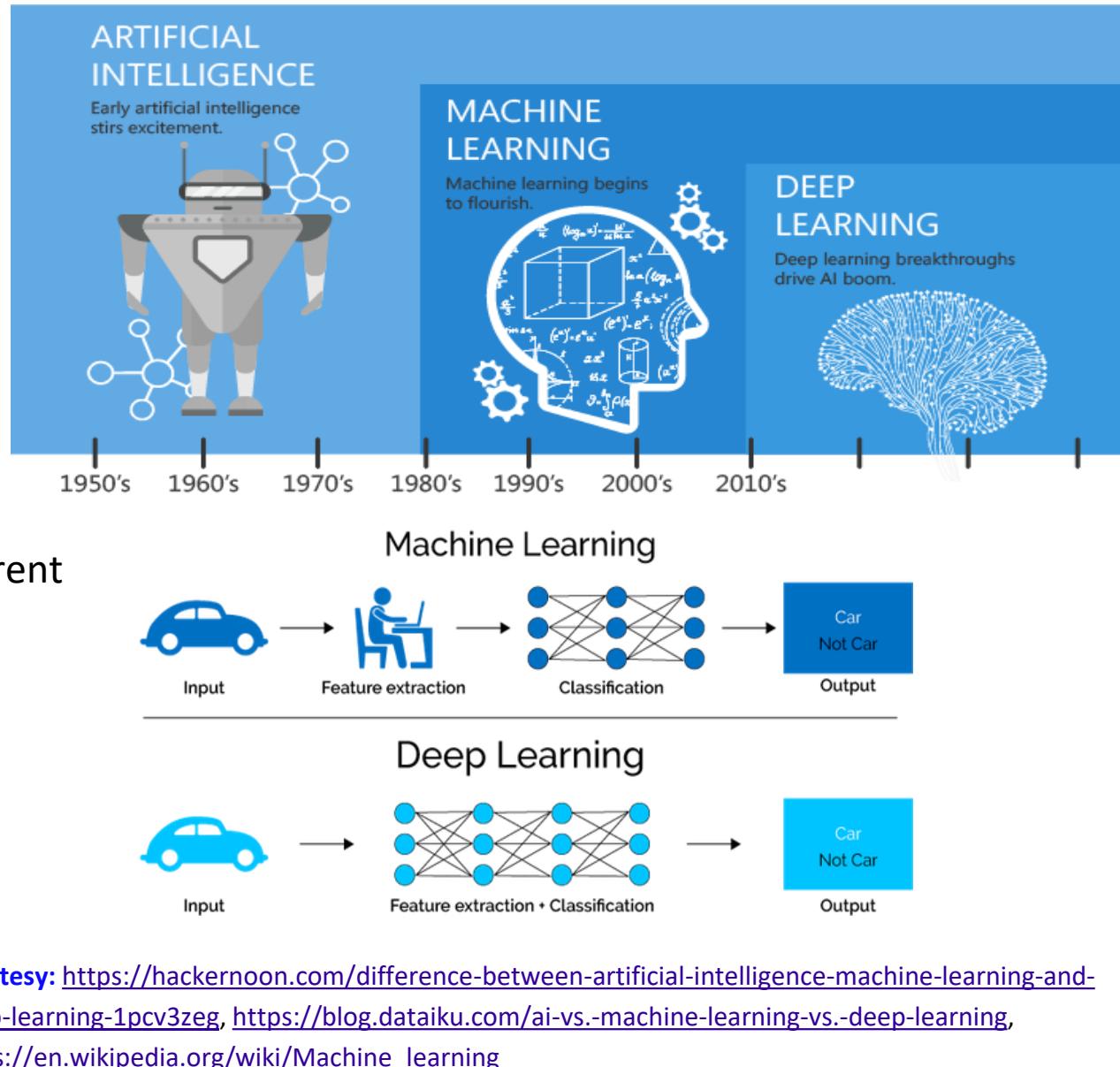
<https://engineering.osu.edu/people/alnaasan.1>

Outline

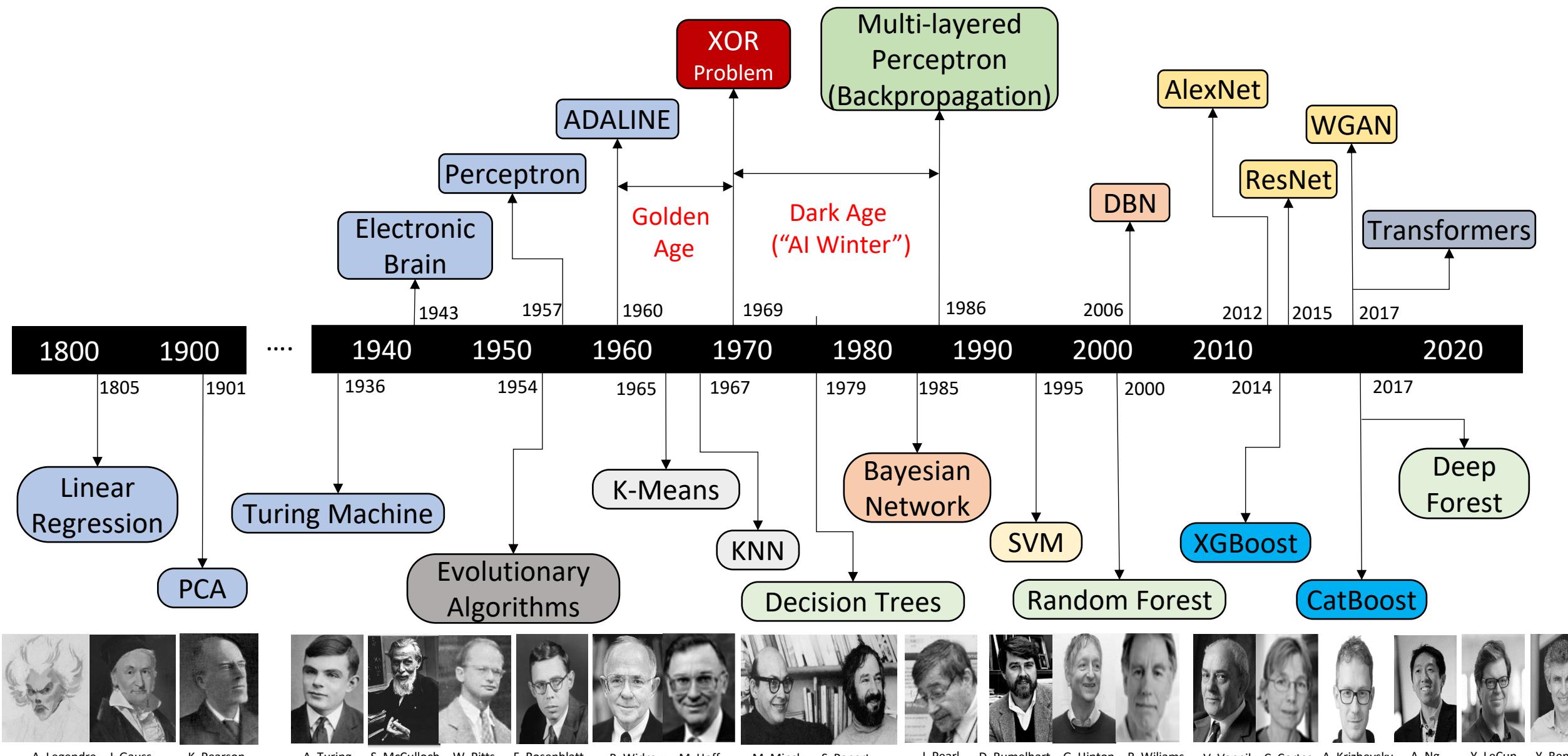
- **Introduction**
 - **The Past, Present, and Future of AI**
 - Machine Learning and Deep Neural Networks
 - Diverse Applications of Deep Learning
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

What is Machine Learning and Deep Learning?

- Machine Learning (ML)
 - “the study of computer algorithms to improve automatically through experience and use of data”
- Deep Learning (DL) – a subset of ML
 - Uses Deep Neural Networks (DNNs)
 - **Perhaps, the most revolutionary subset!**
- Based on learning data representation
- DNN Examples: Convolutional Neural Networks, Recurrent Neural Networks, Hybrid Networks
- Data Scientist or Developer Perspective for using DNNs
 1. Identify DL as solution to a problem
 2. Determine Data Set
 3. Select Deep Learning Algorithm to Use
 4. Use a large data set to train an algorithm



History: Milestones in the Development of ML/DL

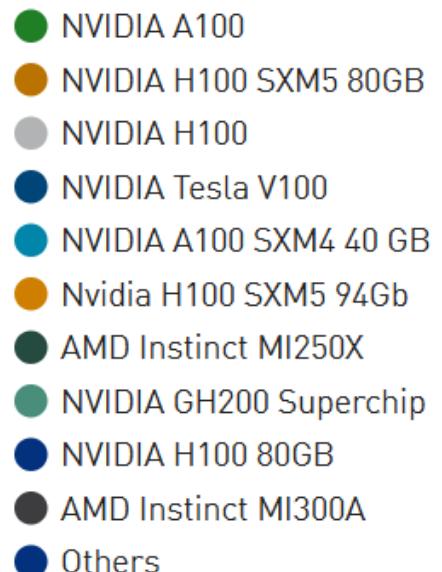
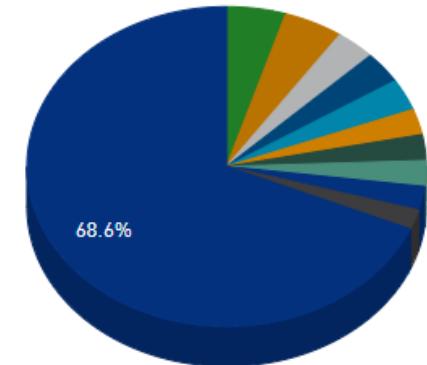


DL and High-Performance Architectures

- NVIDIA GPUs are the main driving force for faster training of DL models
 - The ImageNet Challenge - (ILSVRC) -- 90% of the teams used GPUs (2014)*
 - Kaggle is a community for ML and data science and known for hosting competitions:
 - Provides free GPU access to participants due to wide acceptance by the community
- However, High Performance Architectures for DL and HPC are evolving
 - 215/500 Top HPC systems are using accelerator/co-processor (Jun '25)
 - DGX-1 (Pascal), DGX-2 (Volta), DGX A100, DGX H100, HGX A100, HGX H100
 - Dedicated DL supercomputers
 - NVIDIA Eos – An Exaflop AI Supercomputer using DGX H100 (Announced)
 - AMD Instinct MI300A GPUs power El Capitan – the #1 Top500 hosted at the Lawrence Livermore National Laboratory
 - AMD EPYC (Rome/Milan) CPUs have 64 cores/socket (Frontier – #2 on Top500)
 - Sapphire Rapids Xeon CPUs have 52 cores/socket (Aurora – #3 on Top500)
 - Domain Specific Accelerators for DNNs are also emerging

*<https://blogs.nvidia.com/blog/2014/09/07/imagenet/>

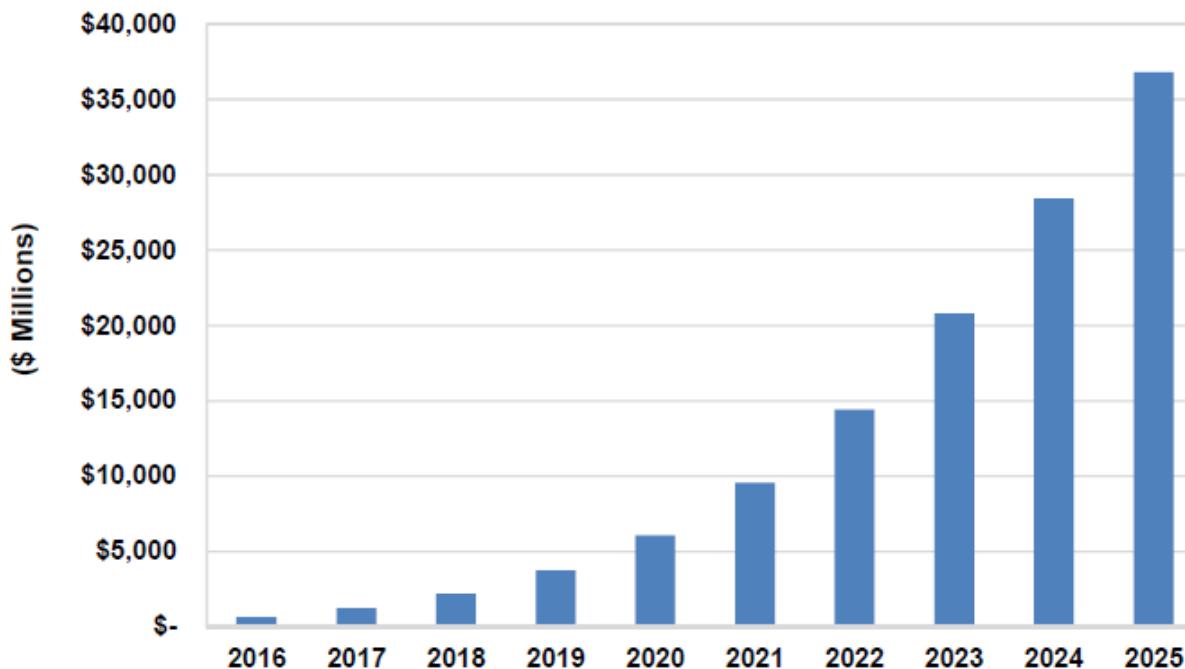
Accelerator/Co-Processor System Share



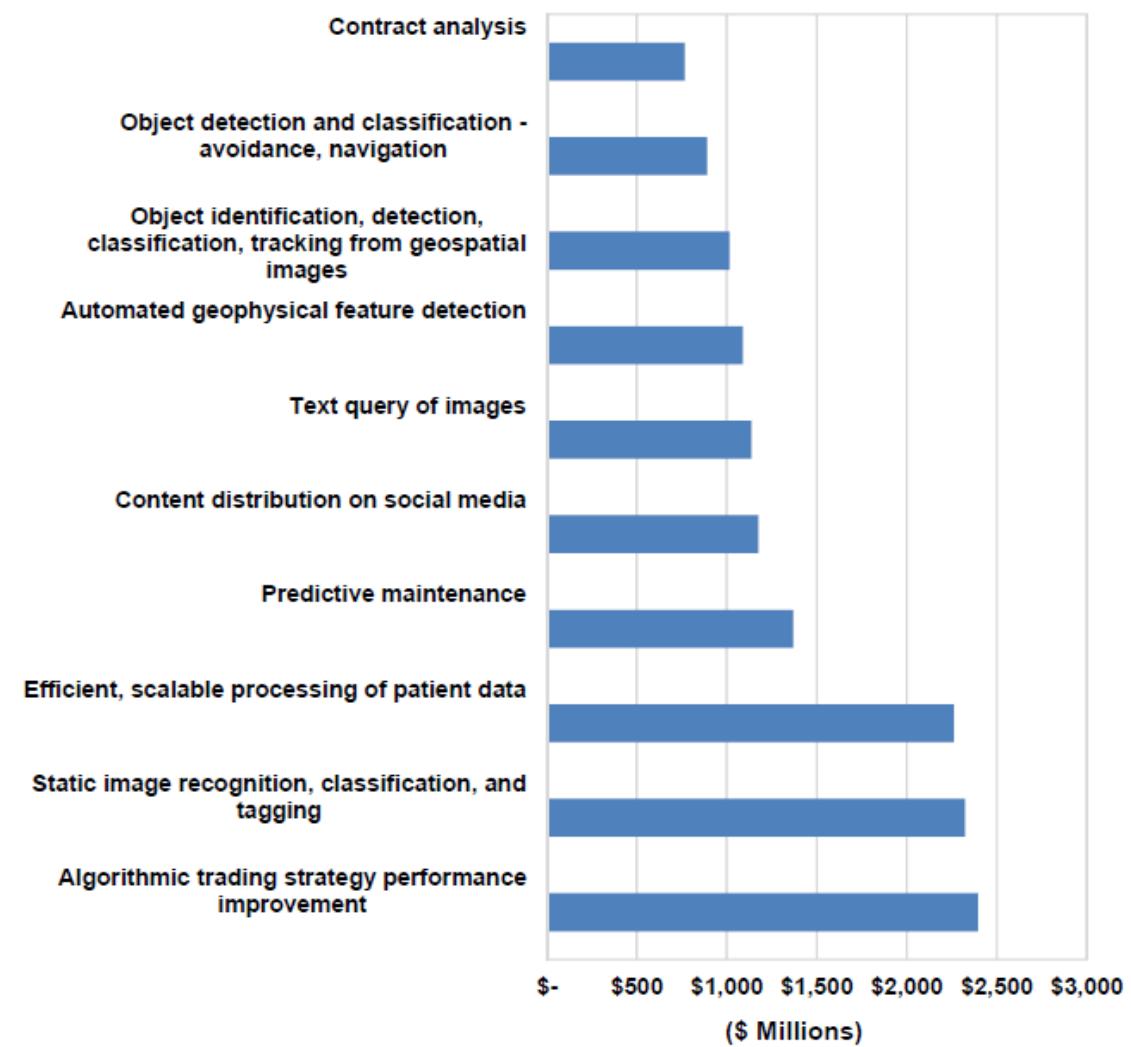
Accelerator/CP
Performance Share
www.top500.org

Artificial Intelligence Use Cases and Growth Trends

1.1 Artificial Intelligence Revenue, World Markets: 2016-2025

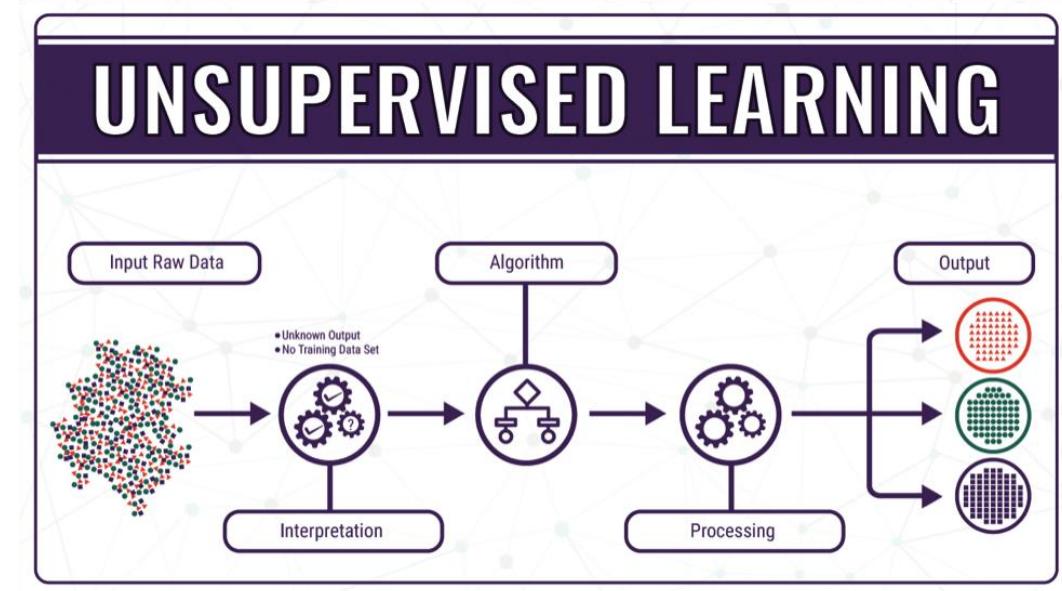
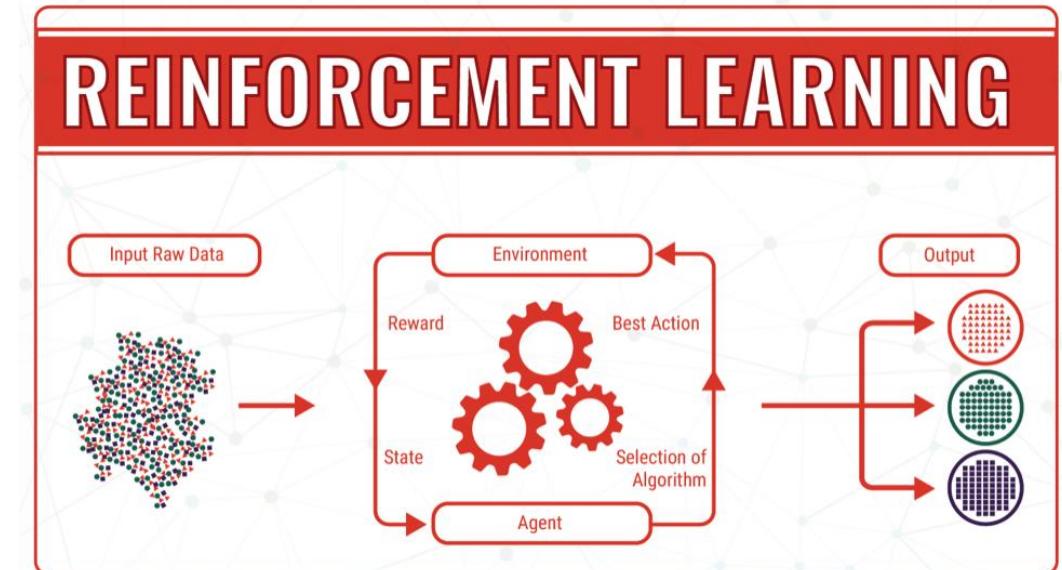
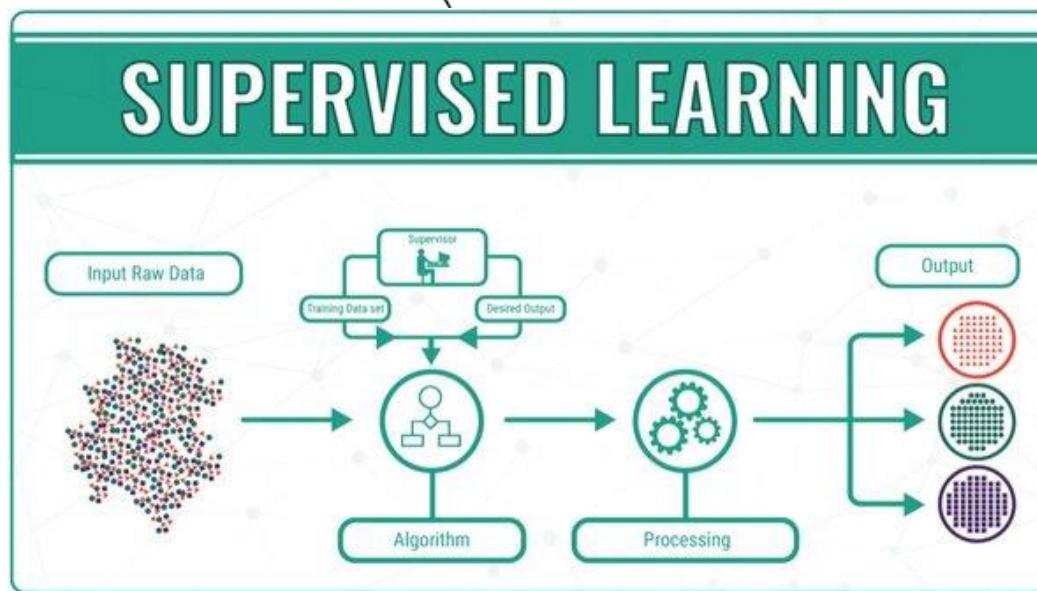
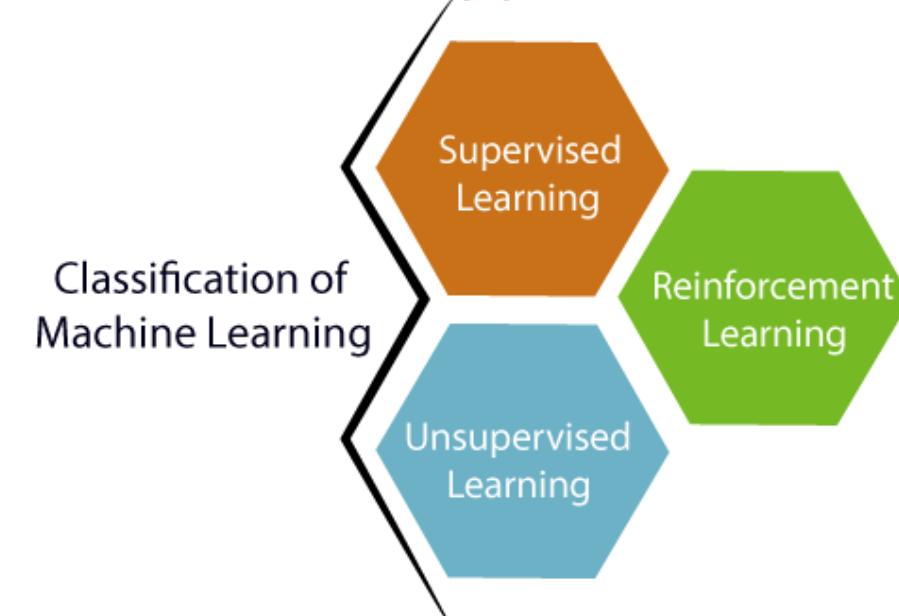


1.2 Artificial Intelligence Revenue, Top 10 Use Cases, World Markets: 2025



Courtesy: <https://www.top500.org/news/market-for-artificial-intelligence-projected-to-hit-36-billion-by-2025/>

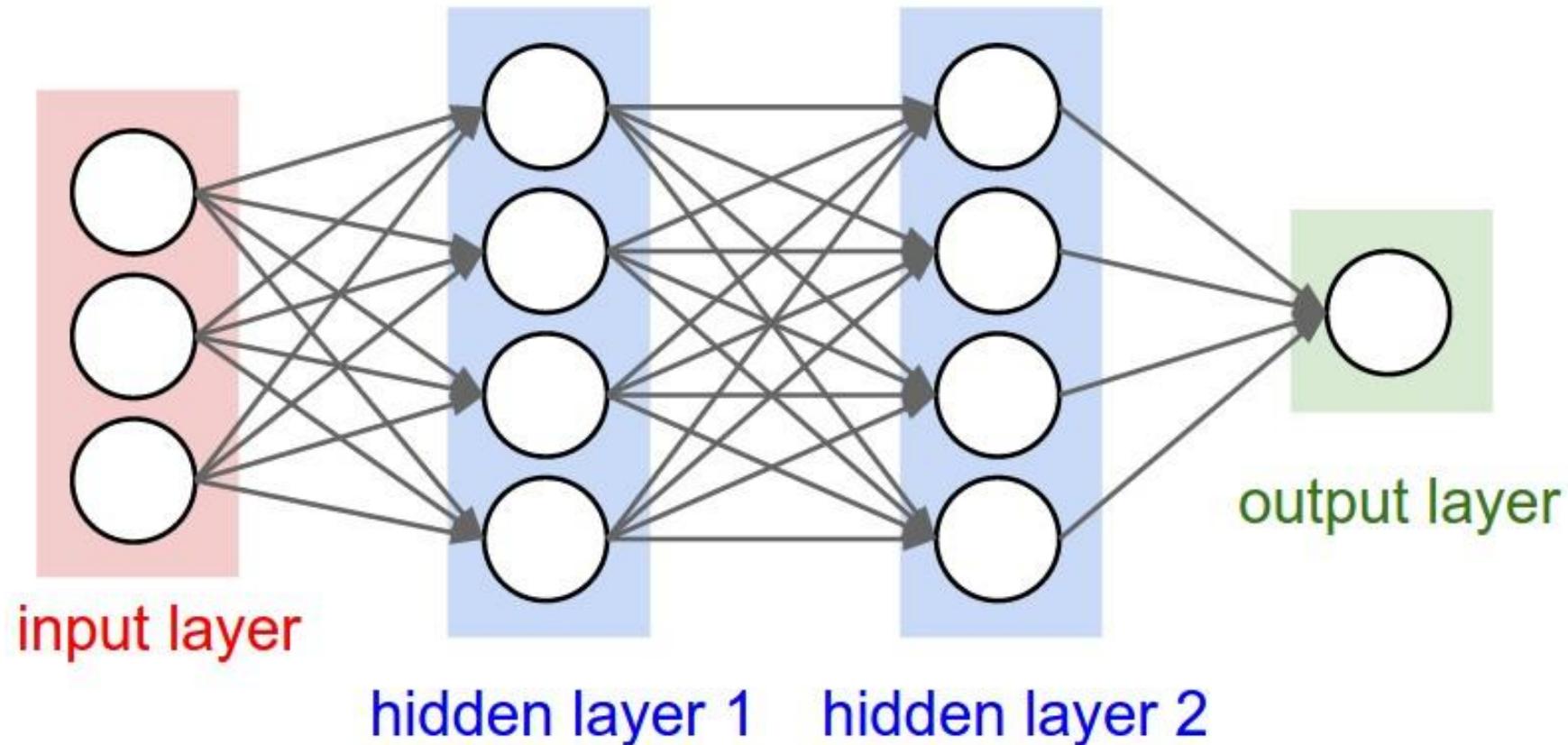
Three Main Types of Machine Learning



Courtesy: <https://bigdata-madesimple.com/machine-learning-explained-understanding-supervised-unsupervised-and-reinforcement-learning/>

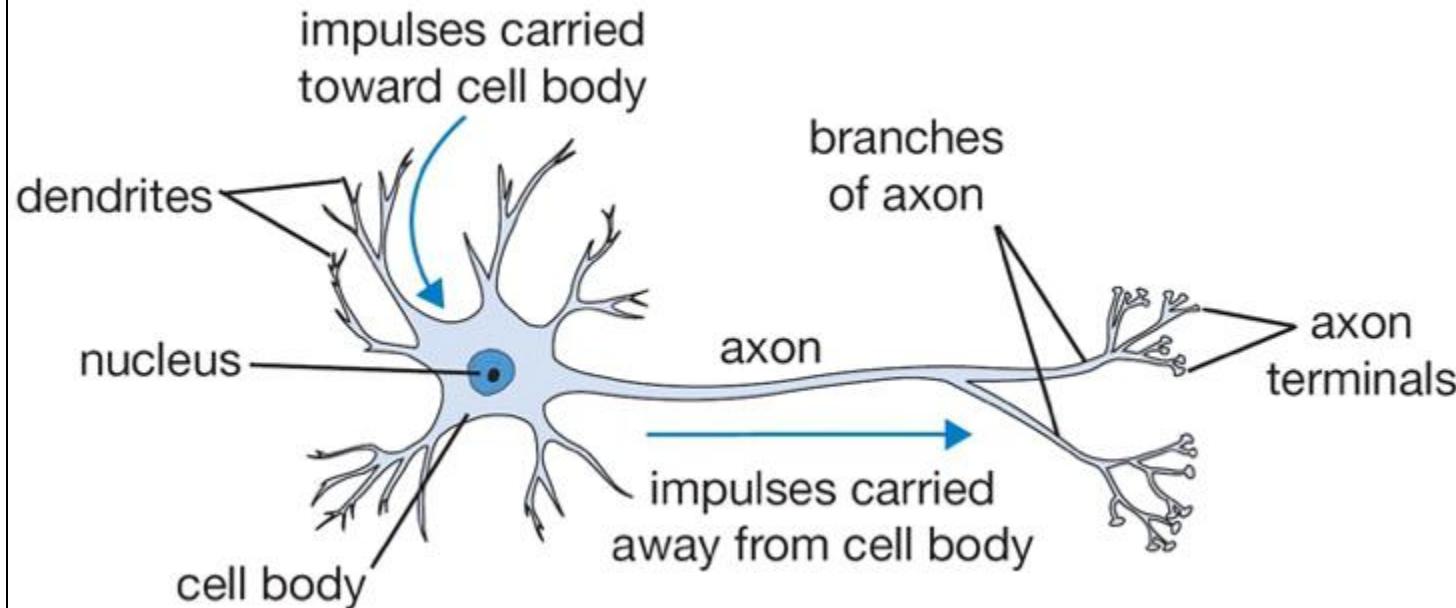
So what is a Deep Neural Network?

- Example of a 3-layer Deep Neural Network (DNN) – (input layer is not counted)

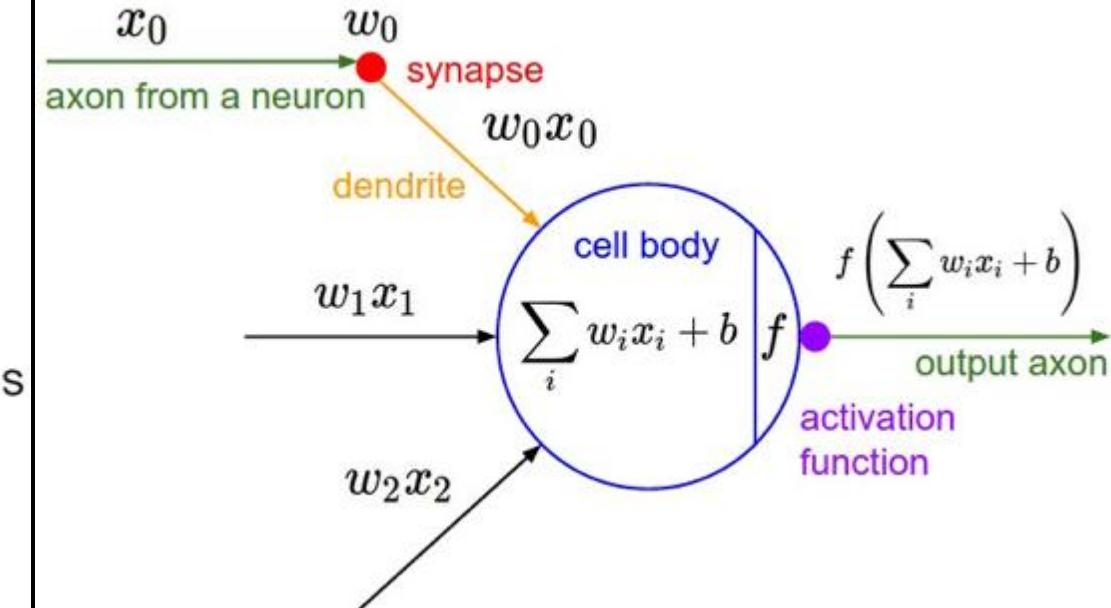


Courtesy: <http://cs231n.github.io/neural-networks-1/>

Graphical/Mathematical Intuitions for DNNs



Drawing of a Biological Neuron

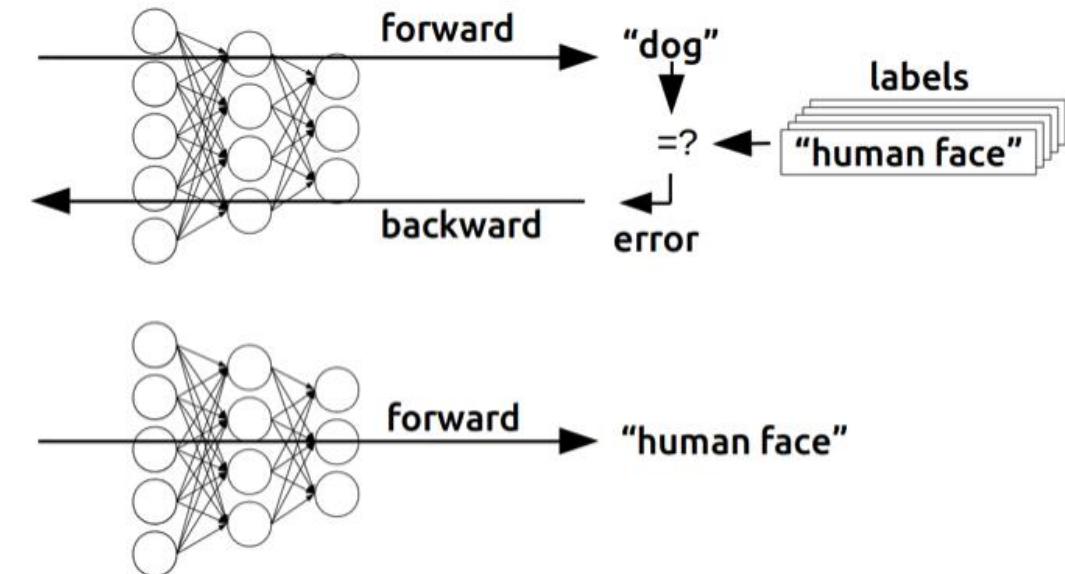
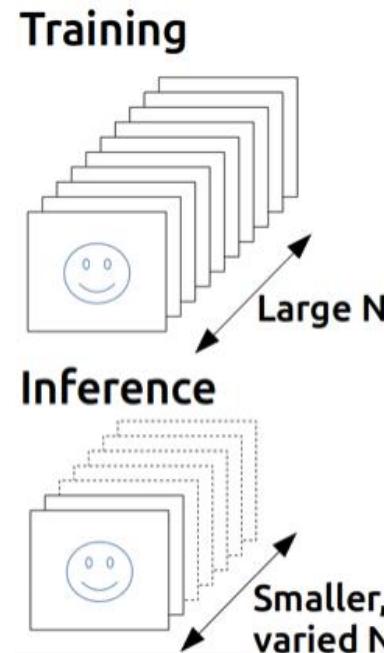


The Mathematical Model

Courtesy: <http://cs231n.github.io/neural-networks-1/>

Key Phases of Deep Learning

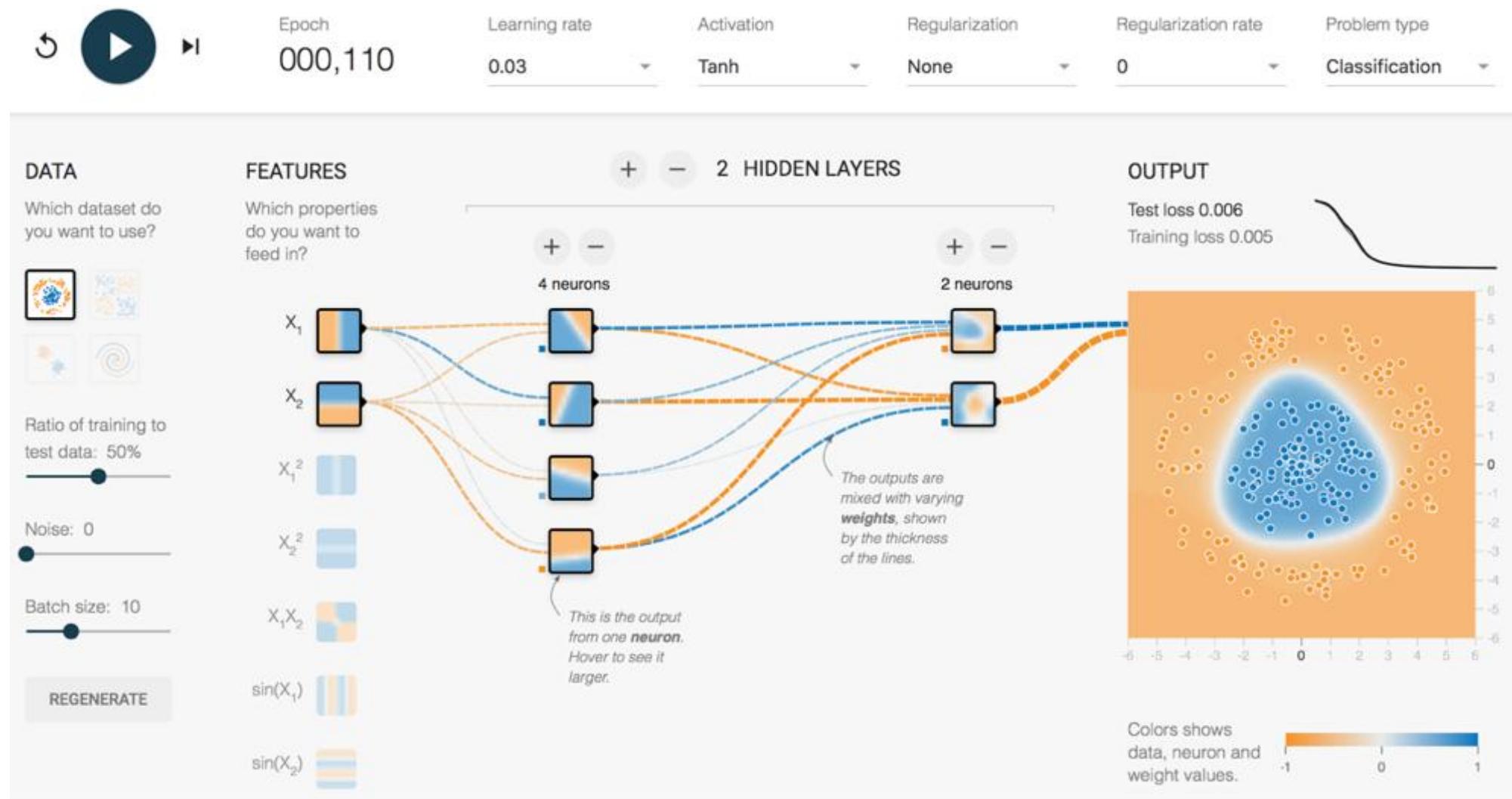
- Training is compute intensive
 - Many passes over data
 - Can take days to weeks
 - Model adjustment is done
- Inference
 - Single pass over the data
 - Should take seconds
 - No model adjustment
- Challenge: How to make “**Training**” faster?
 - Need Parallel and Distributed Training...



Courtesy: <https://devblogs.nvidia.com/>

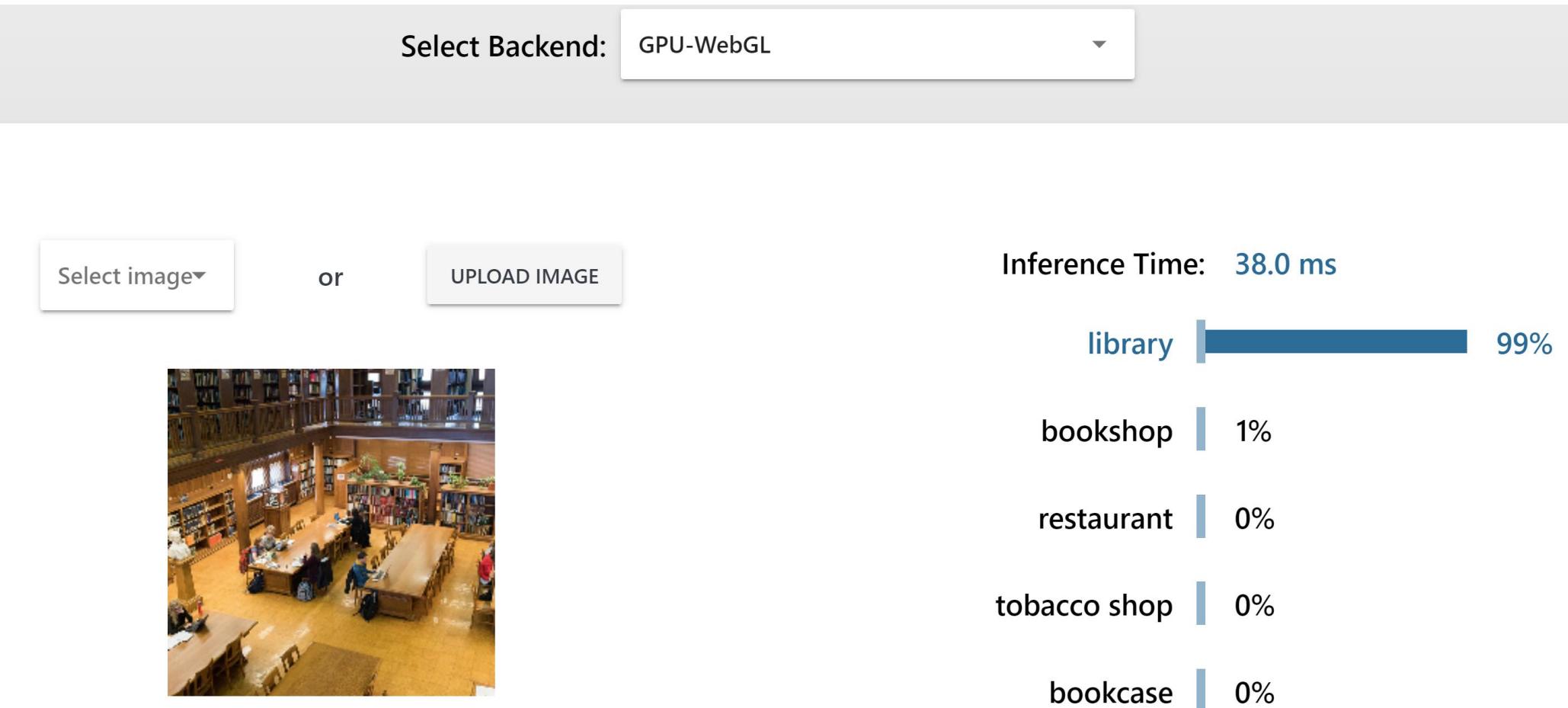
TensorFlow playground (Quick Demo)

- To actually train a network, please visit: <http://playground.tensorflow.org>

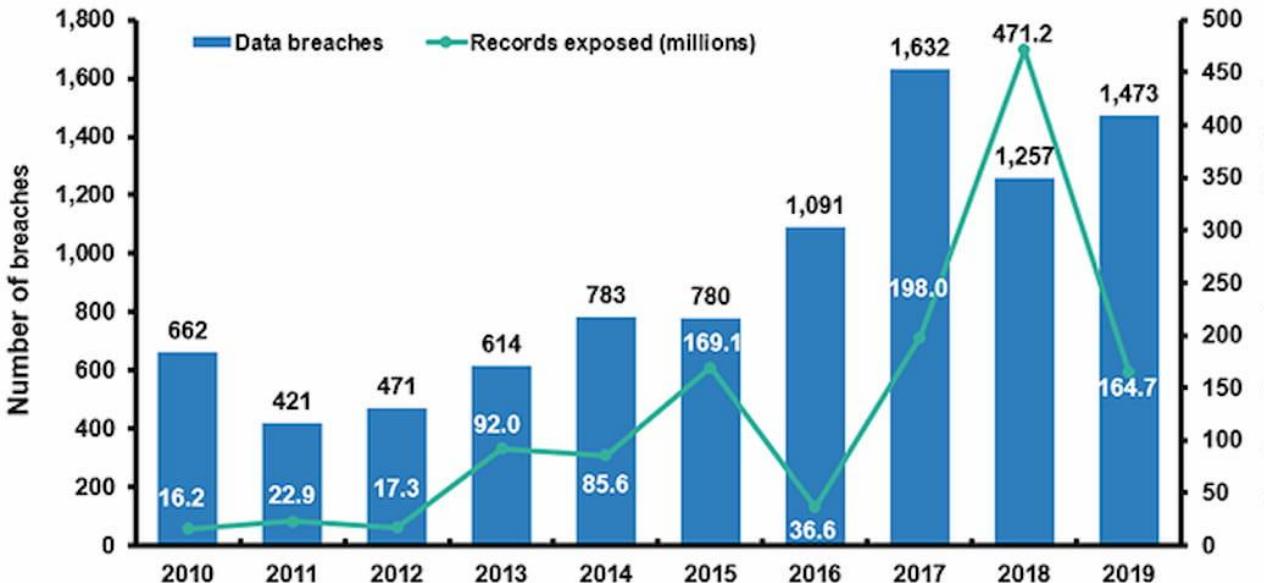


Inference on trained ResNet50 (Quick Demo)

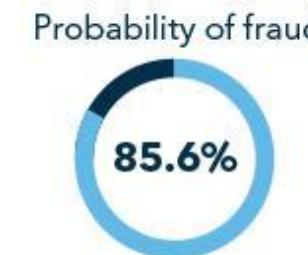
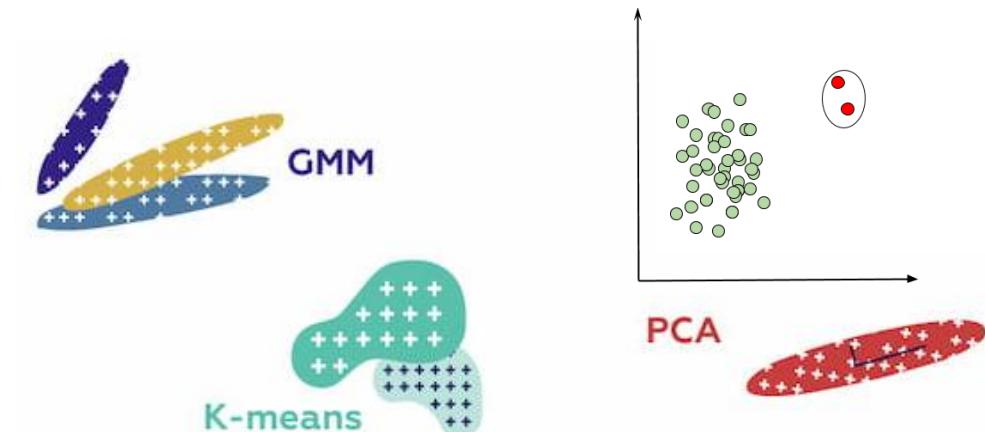
- To try your own image, please visit: <https://microsoft.github.io/onnxjs-demo/#/resnet50>



Credit Card Fraud Detection using Unsupervised Techniques



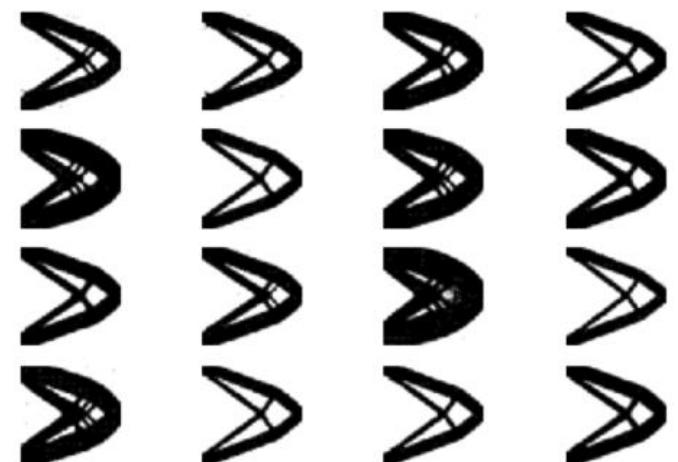
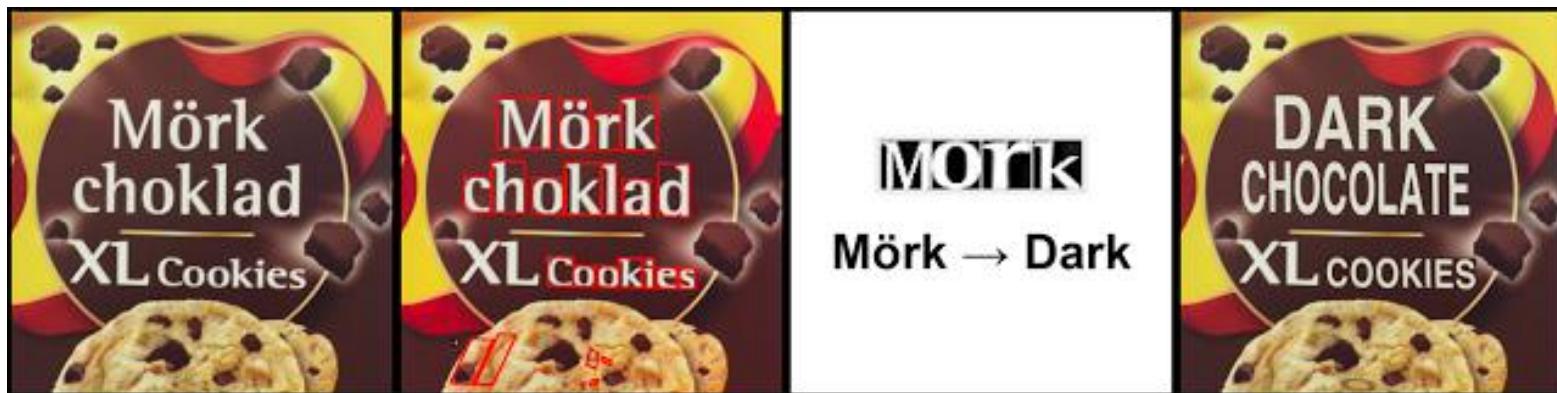
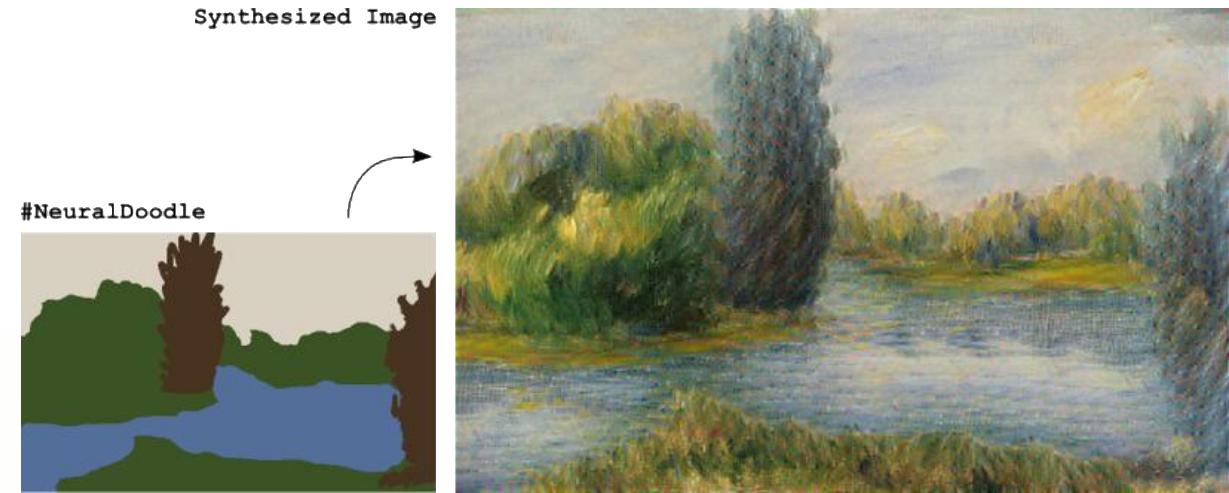
... almost \$112 million due to credit card fraud in 2019.



Courtesy: <https://spd.group/machine-learning/fraud-detection-with-machine-learning>

https://www.sas.com/en_us/insights/articles/risk-fraud/fraud-detection-machine-learning.html

The Impact of Deep Learning on Application Areas



Google Translate



Courtesy: <https://www.theverge.com/2015/1/14/7544919/google-translate-update-real-time-signs-conversations>

Self Driving Cars



Courtesy: <http://www.teslarati.com/teslas-full-self-driving-capability-arrive-3-months-definitely-6-months-says-musk/>

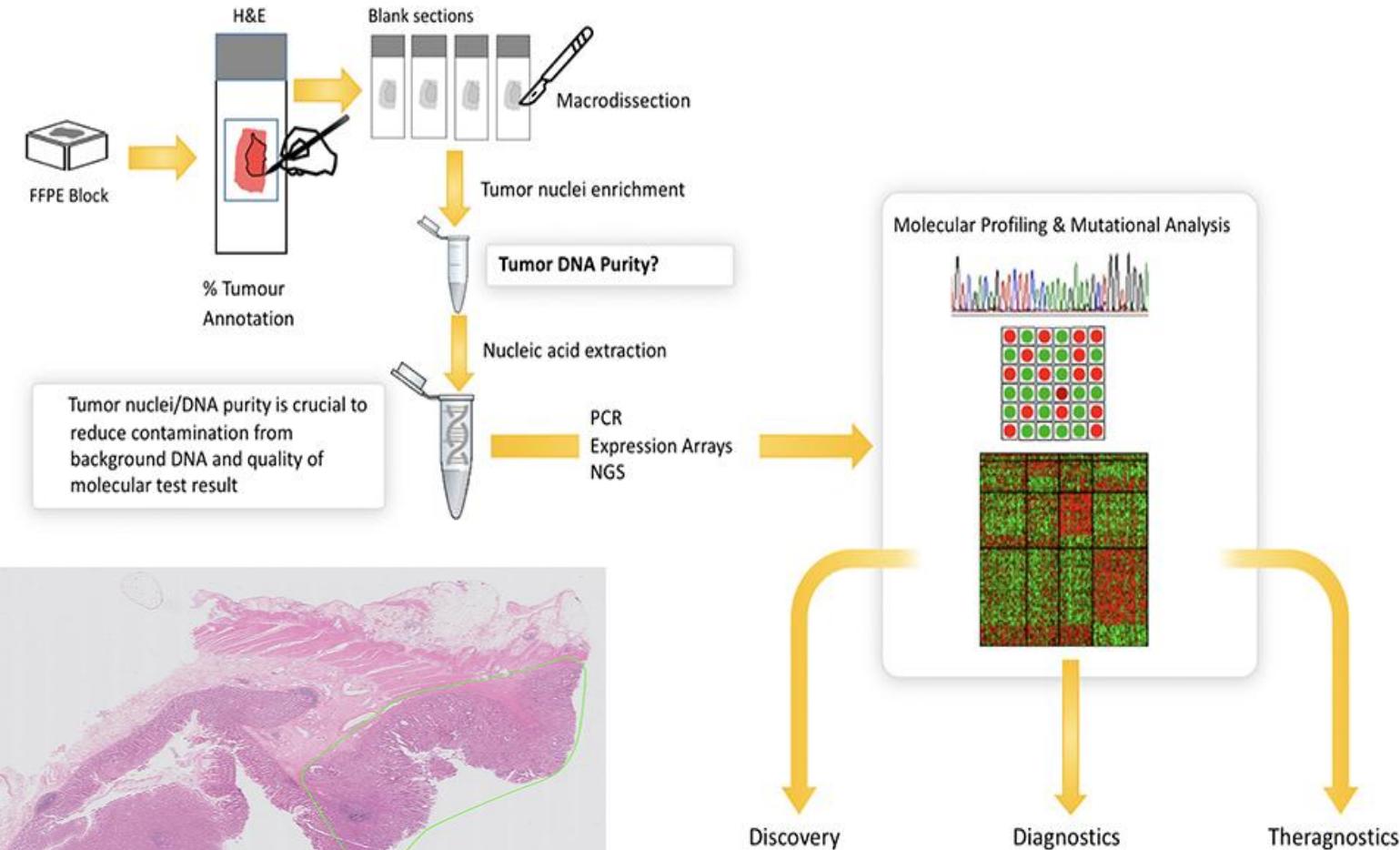
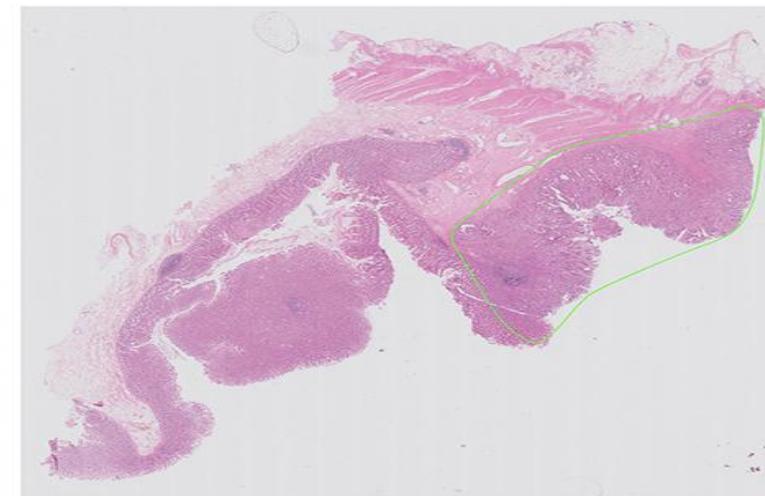
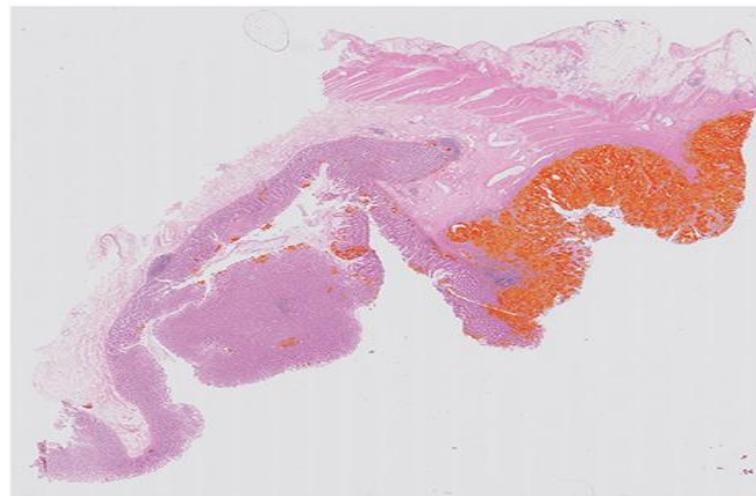
Food/Coffee Distribution in OSU Campus



Will have significant impact in distribution of groceries, food, packages, mails, etc.

AI-Driven Digital Pathology

- Applications
 - Prostate Cancer Detection
 - Metastasis Detection in Breast Cancer
 - Genetic Mutation Prediction
 - Tumor Detection for Molecular Analysis



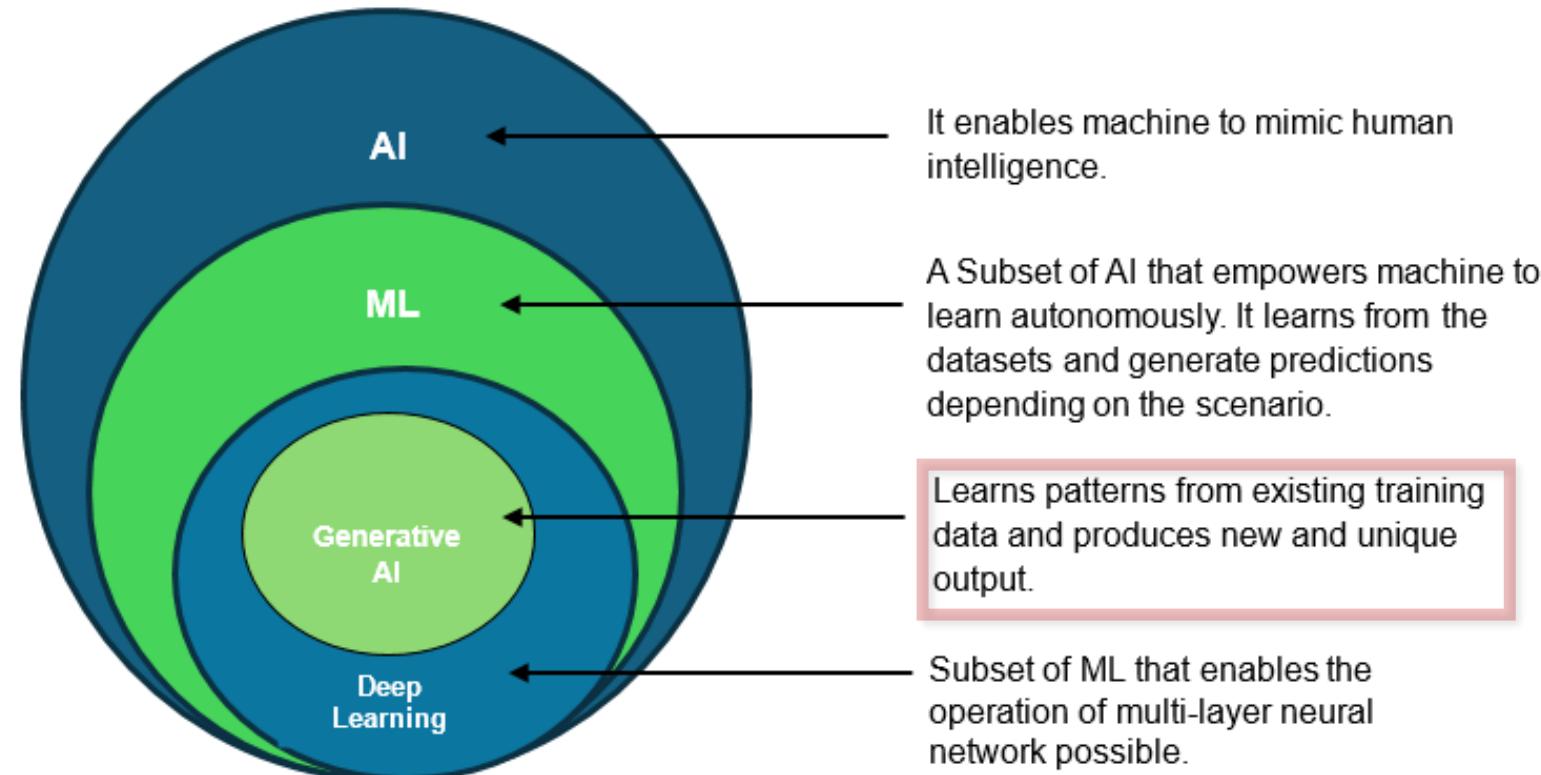
Courtesy: <https://www.frontiersin.org/articles/10.3389/fmed.2019.00185/full>

What is Generative AI?

- Generative AI is a subset of Deep Learning which creates new content like text, images, videos, or audio based on the data it was trained on.

- Examples:
 - Text: GPT, LLaMA, and DeepSeek.
 - Images: DALL-E and Stable Diffusion.
 - Videos: Runway and Sora.
 - Audio: AudioPaLM and VALL-E.

- What is not Generative AI?
 - Discriminative models that perform:
 - Classification
 - Regression
 - Object detection
 - Clustering
 - etc.



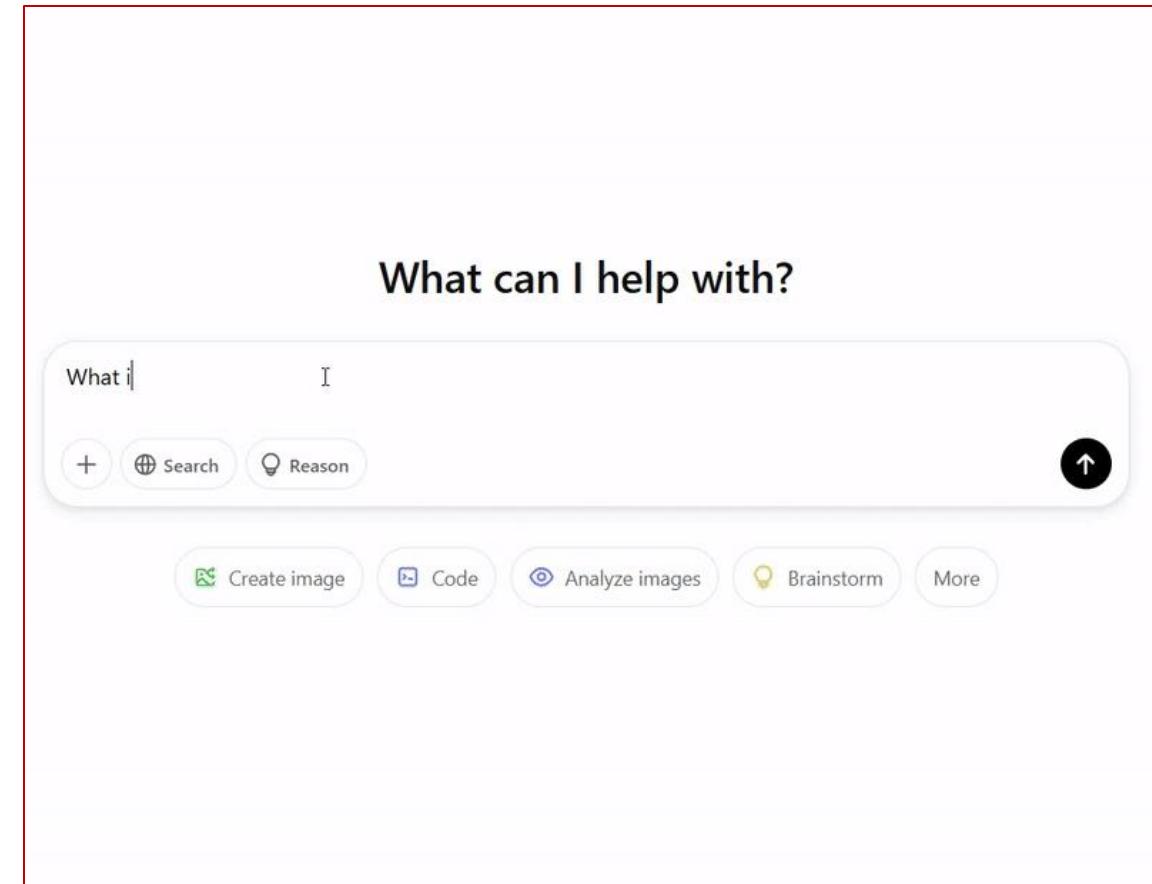
Courtesy: <https://www.tutorialspoint.com/gen-ai/ml-and-generative-ai.htm>

Generative AI – Inference

In inference, the model generates outputs based on input prompts. For autoregressive models (most LLMs), inference follows an **iterative loop**, where each generated token (word) is **fed back** as input for the next step until completion.

LLM inference requires low-latency, high-throughput compute with the following key QoS (Quality of Service) requirements:

- **Low Latency** – Ensures fast response times, crucial for interactive applications.
- **Efficient Batch Processing** – Optimized for serving multiple queries in parallel to maximize throughput.
- **Mixed-Precision Support (FP16/BF16/INT8)** – Reduces compute overhead while maintaining accuracy.
- **High-Speed Interconnects (NVLink, InfiniBand)** – Required for multi-GPU inference to minimize communication bottlenecks.
- **High Memory Bandwidth** – To efficiently load large model weights and handle activation memory.



Online LLM Inferencing

Outline

- Introduction
- **Deep Learning Frameworks**
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

Beginnings of DL Frameworks – Identifying Cats!

≡ WIRED BACKCHANNEL BUSINESS CULTURE GEAR IDEAS SCIENCE SECURITY

WIRED STAFF SCIENCE JUN 26, 2012 11:15 AM

Google's Artificial Brain Learns to Find Cat Videos

When computer scientists at Google's mysterious X lab built a neural network of 16,000 computer processors with YouTube, it did what many web users might do -- it began to look for cats.



ZDNET

tomorrow
belongs to those who embrace it
today

trending innovation home & office business finance

/ innovation

Home / Innovation

Google brain simulator teaches cats to recognize cats

A neural network of 16,000 computers was let loose on YouTube images for three days. What did it learn by the end? How to recognize cats.



Written by Laura Shin, Contributor on June 26, 2012

in

/ must read



The HP Elite Dragonfly Chromebook has no business being this good

Read now ↗



The New York Times

How Many Computers to Identify a Cat? 16,000

[Give this article](#) [Email](#) [Print](#)

An image of a cat that a neural network taught itself to recognize. Jim Wilson/The New York Times

By John Markoff

June 25, 2012

MOUNTAIN VIEW, Calif. — Inside Google's secretive X laboratory,

[MOBILE](#) [HEXUS.TV](#) [COMPETITIONS](#) [FORUMS](#)

[Like](#)

10

Network can identify a cat

» began working on a simulation of the human brain. To test the system, the researchers presented it with images, which may or may not contain faces, at various sizes. The system then learned to recognize higher-level concepts such as cat faces and human bodies.

with 1,000 machines (16,000 cores"). The researchers say the system can learn to identify a cat by analyzing 20,000 object categories from ImageNet, a leap of 70% over previous work. Also the system was set loose on 10 million random 200x200 pixel images. The system successfully identified and grouped cat faces.

After scientist Andrew Y. Ng and the Google fellow Jeff Dean. Dr. Ng said, "This is a breakthrough in computer vision. It's a major step forward in training, 'This is a cat,' it basically invented the concept of a cat. We've taken a page from the biological brain." Dr. Ng added, "The idea is that instead of having teams of researchers come up with features and then train the algorithm and you let the data speak and have the algorithm learn, we've taken a page from the biological brain and said, 'Let's just give the algorithm lots of data and let it figure out what's important.' It's based on the idea that the brain is much more like how biological brains learn recognition.

Beginnings of DL Frameworks – Identifying Cats!

- Done at the secretive X lab at Google
- Neural network of 16,000 computer processors with 1 billion connections
- This network browsed YouTube and began to look for cats
- Dataset:
 - 10 million randomly selected YouTube video thumbnails
 - 20, 000 different items
- This brain achieved 81.7% accuracy in detecting human faces, 76.7% accuracy in identifying human parts, and 74.8% accuracy for cats
- Utilized model parallelism
- This work is described in detail in “Building High-Level Features Using Large Scale Unsupervised Learning”

[1] <https://arxiv.org/pdf/1112.6209.pdf>

Beginnings of DL Frameworks – DL with COTS HPC

- An influential paper “Deep Learning with COTS HPC systems” was published in ICML ’13 (<http://proceedings.mlr.press/v28/coates13.pdf>)
- The paper solves a similar problem as “identifying cats” but relies on GPUs and **MVAPICH** for communication:
 - 6.5 times larger model than state-of-the-art in few days with 2% of the original machines
 - Neural networks of DistBelief scale can be trained with 3 machines
- Hardware:
 - A cluster of GPU servers with InfiniBand interconnect
- Software:
 - Custom CUDA kernels for matrix-vector and matrix-matrix operations
 - MVAPICH2-GDR was used as the MPI library for communicating data between GPUs
- Most importantly, this project formed the basis of the cuDNN project at NVIDIA

The NVIDIA cuDNN Library

- cuDNN is a GPU-accelerated library of primitives for DNNs
- cuDNN provides optimized and efficient routines for:
 - Forward and backward convolution
 - Pooling
 - Normalization
 - Activation Layers
- cuDNN Accelerated DL Frameworks

Caffe

 Caffe2



Chainer



 mxnet

 PaddlePaddle

 PyTorch

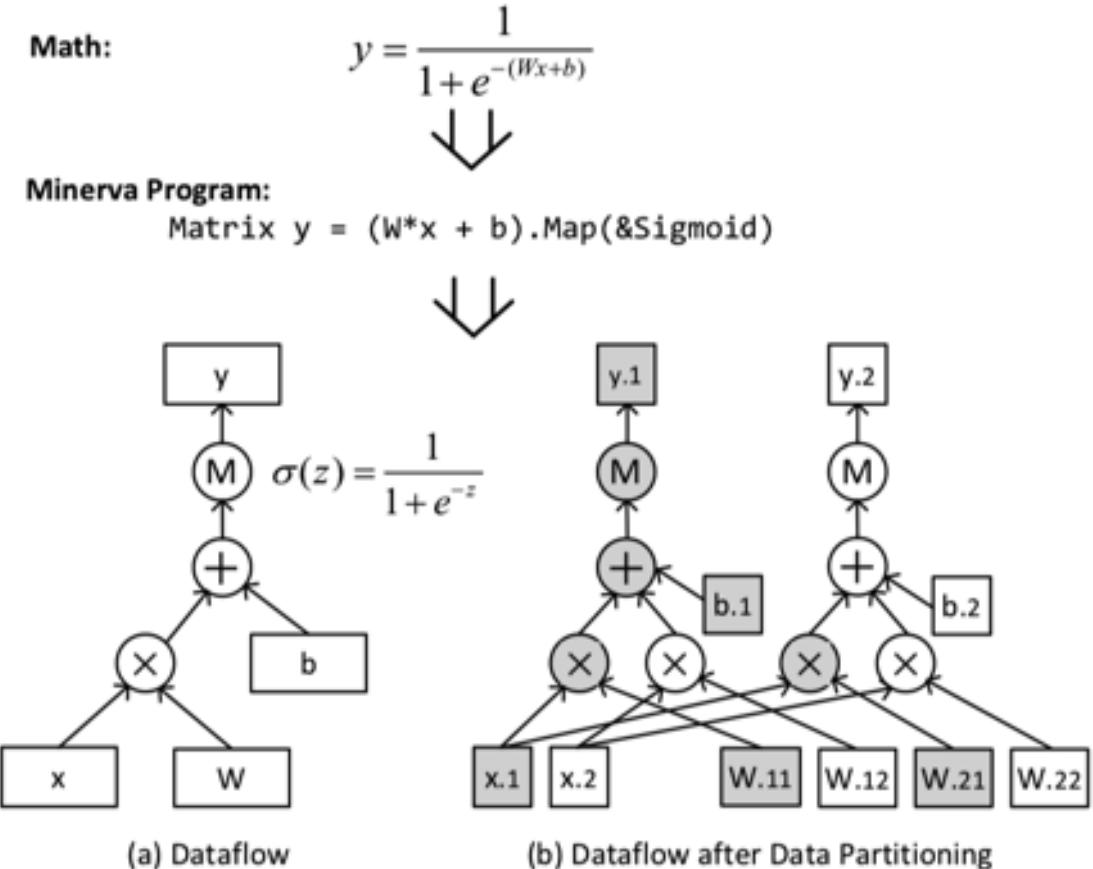
 TensorFlow

 torch

 Wolfram
Language™

DL Frameworks, Hardware Architectures, and Distributed Training

- Main objectives of DL frameworks:
 - Hide complex mathematics
 - Allow users to focus on DL models
- Support for Parallelism:
 - We have saturated the peak potential of current-generation architectures
 - A single GPU or a many-core CPU is not enough!
- Two strategies to deal with current limitations
 - Parallel (multiple units in a single node) and/or Distributed (multiple nodes) training of DNNs
 - Dedicated hardware architectures for DNNs are being developed (TPUs, Graphcore , etc.)



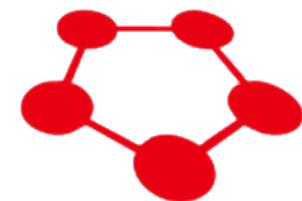
Statement and its dataflow fragment. The data and computing vertexes with different colors reside on different processes.

Courtesy: <https://web.stanford.edu/~rezab/nips2014workshop/submits/minerva.pdf>

Deep Learning Frameworks

- Many Deep Learning frameworks
 - Google TensorFlow
 - Facebook Torch/PyTorch
 - Berkeley Caffe
 - Microsoft CNTK
 - Chainer/ChainerMN
 - Intel Neon/Nervana Graph
- Open Neural Net eXchange (ONNX) Format

Caffe



PYTORCH

PyTorch – Background and History



- PyTorch is a Python adaptation of Torch (written in Lua)
 - Released in 2016 and has gained a lot of traction
- Several contributors and mainly backed by Meta
- Key selling point is ease of expression and “define-by-run” approach
- Build upon previous frameworks like Chainer, Lua Torch, and HIPS
- Originally a Python library but has been moved to C++/C
- Port of Torch framework into Python
- Support for GPU acceleration
- Integration with Numpy
- Automatically generated computational graphs
- Automatic differentiation

Many Other DL Frameworks...

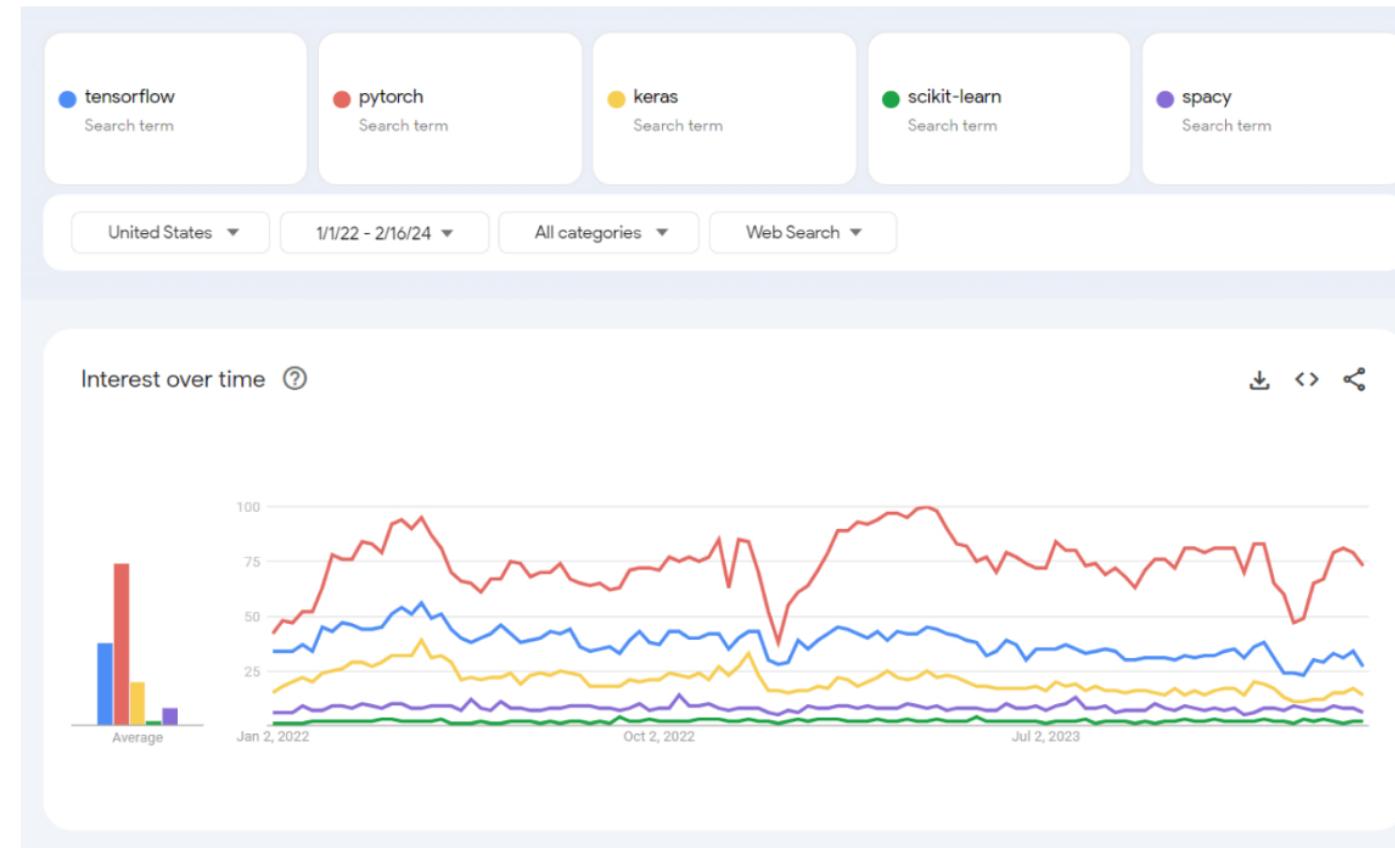
- Caffe – <https://caffe.berkeleyvision.org>
- Keras - <https://keras.io>
- Theano - <http://deeplearning.net/software/theano/>
- Blocks - <https://blocks.readthedocs.io/en/latest/>
- Intel BigDL - <https://software.intel.com/en-us/articles/bigdl-distributed-deep-learning-on-apache-spark>
- The list keeps growing and the names keep getting longer
 - Livermore Big Artificial Neural Network Toolkit (LBANN) -
<https://github.com/LLNL/lbann>
 - Deep Scalable Sparse Tensor Network Engine (DSSTNE) -
<https://github.com/amzn/amazon-dsstne>

Statistics about ML/DL Frameworks

- AI Index report offers very detailed trends about AI and ML
 - Interesting stats. about DL frameworks
- The Gradient* reported in 2019 on *PyTorch winning over TensorFlow* in CVPR, ICML, ICLR and other conferences

* <https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/>

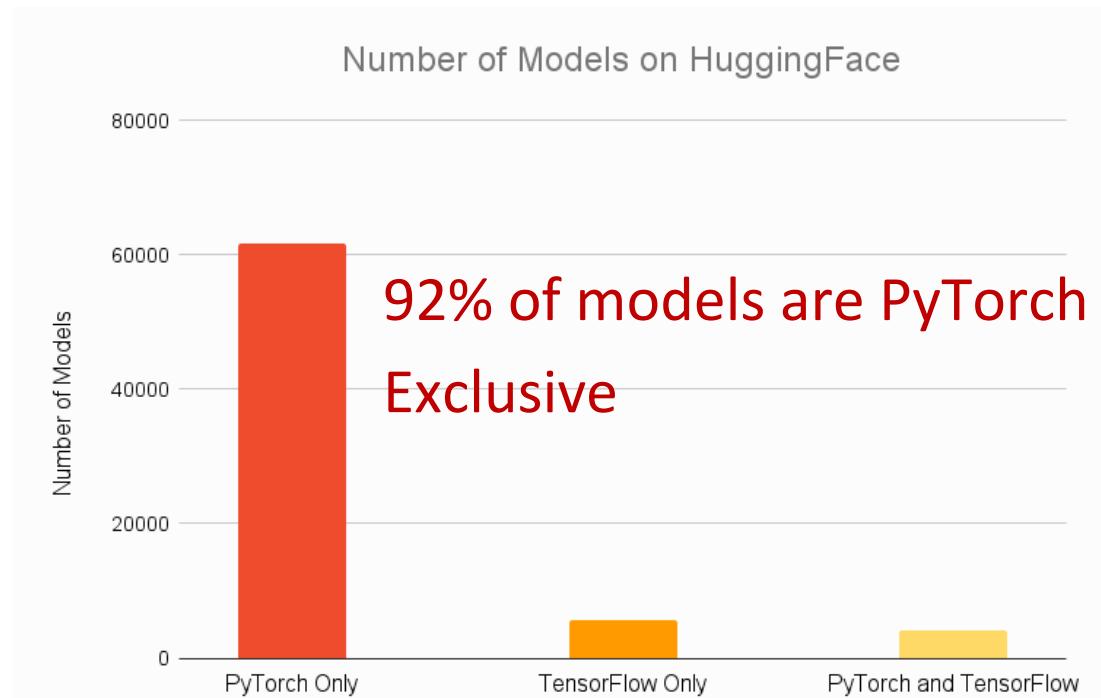
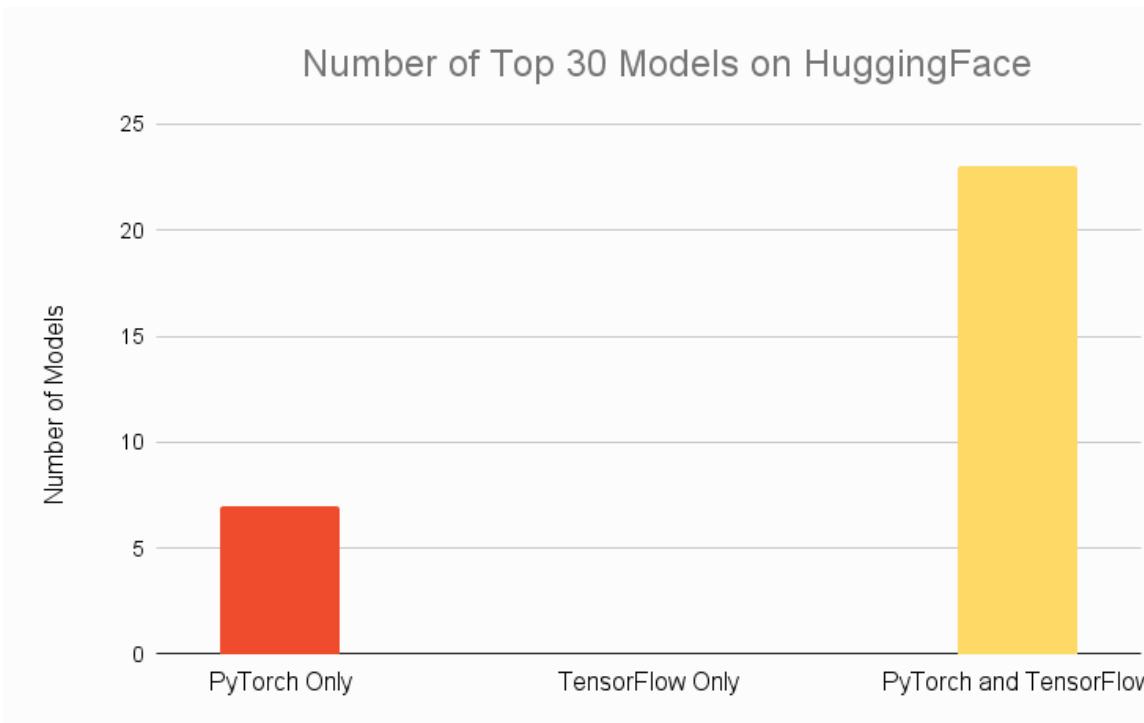
Top 5 fundamental open-source AI frameworks: search trends



Courtesy: <https://clockwise.software/blog/artificial-intelligence-framework/>

PyTorch vs. TensorFlow: Model Availability

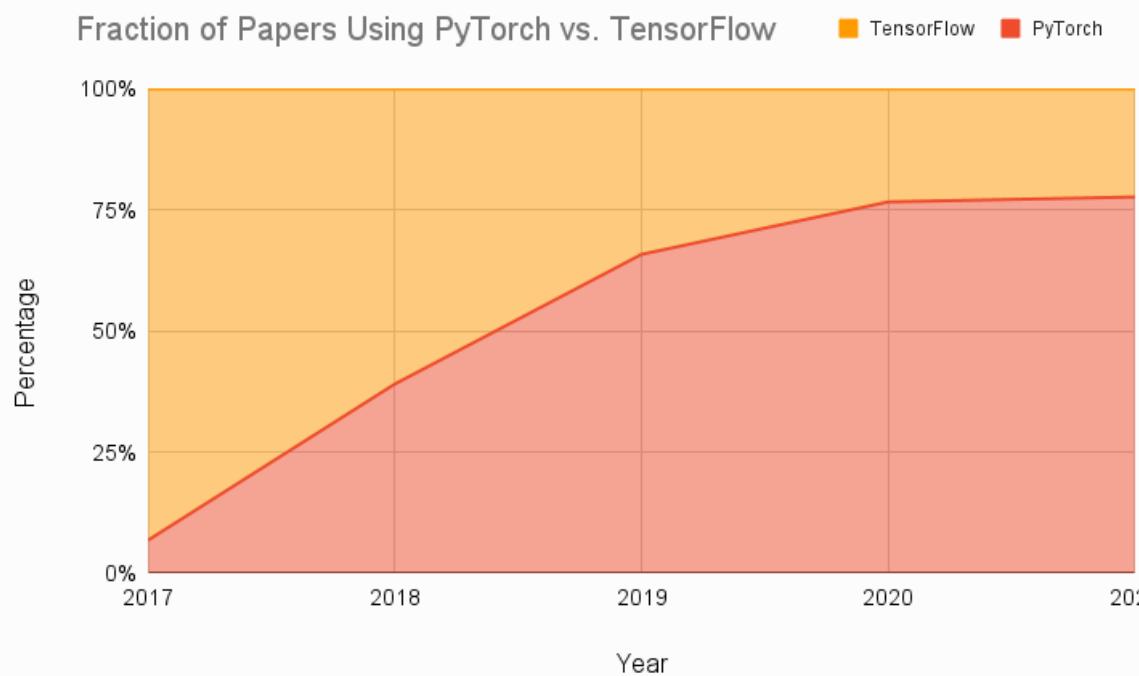
- HuggingFace is a model repository for trained and tuned SOTA models



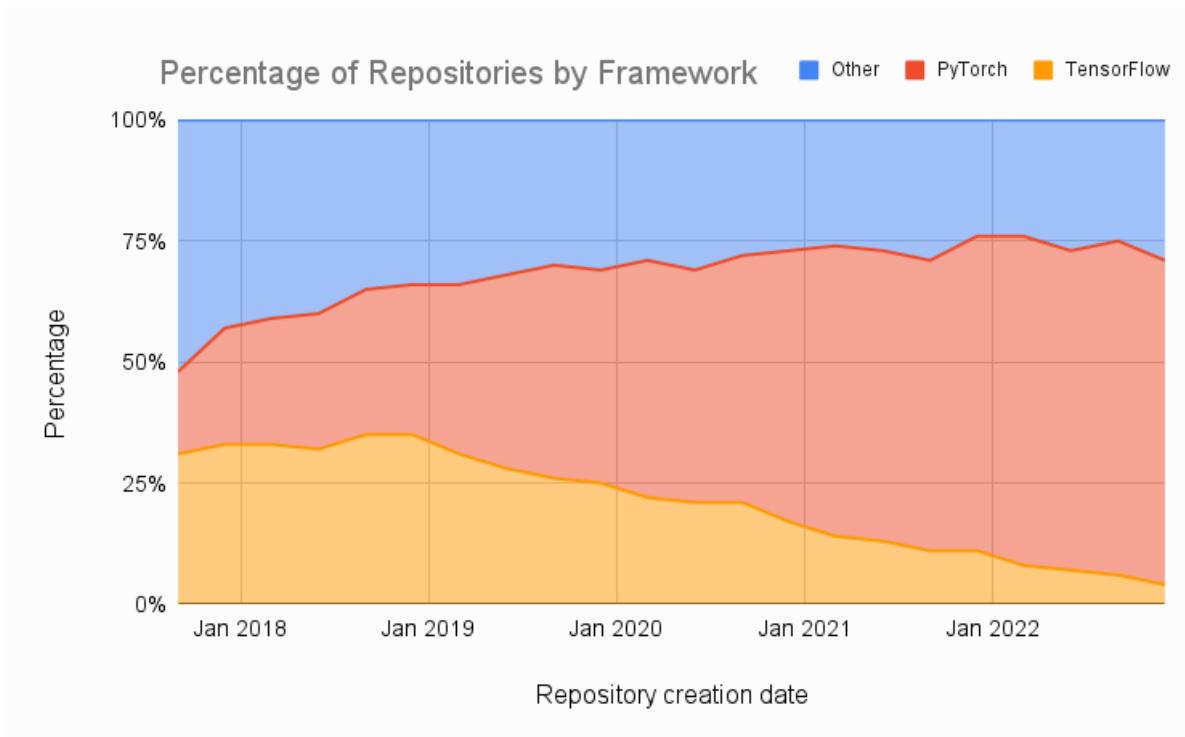
Courtesy: <https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/>

PyTorch vs. TensorFlow: Research Papers & Papers with Code

PyTorch adoption grew from 7%
(in 2017) to 80% (2021)



70% -> PyTorch repositories, 4% ->
TensorFlow repositories (latest quarter)



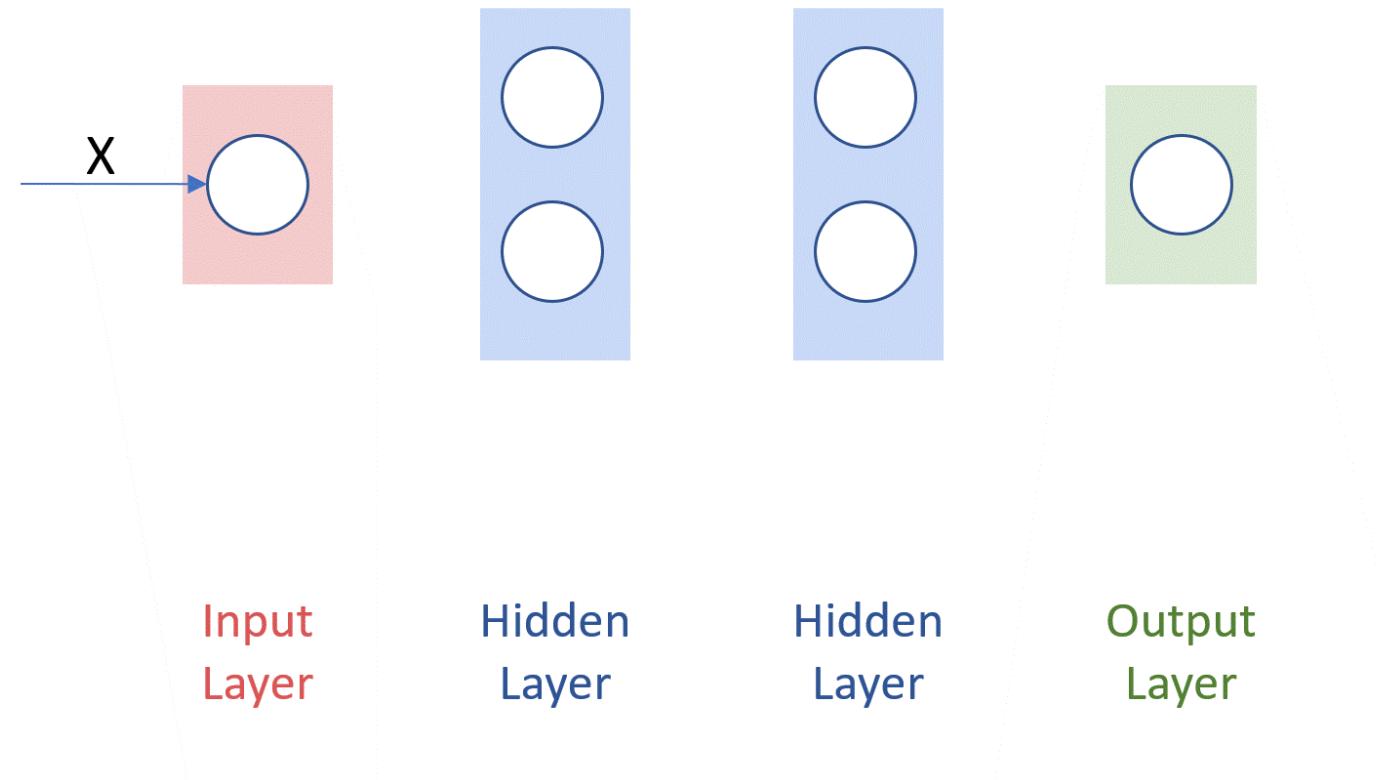
Courtesy: <https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/>

Outline

- Introduction
- Deep Learning Frameworks
- **Deep Neural Network Training**
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

Understanding the Deep Neural Network Concepts

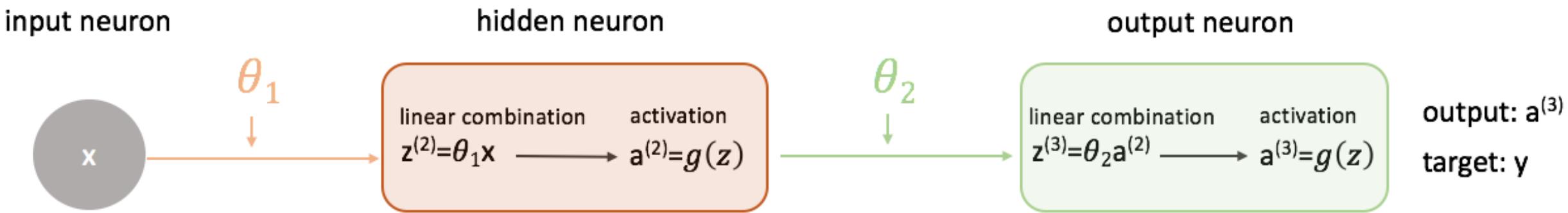
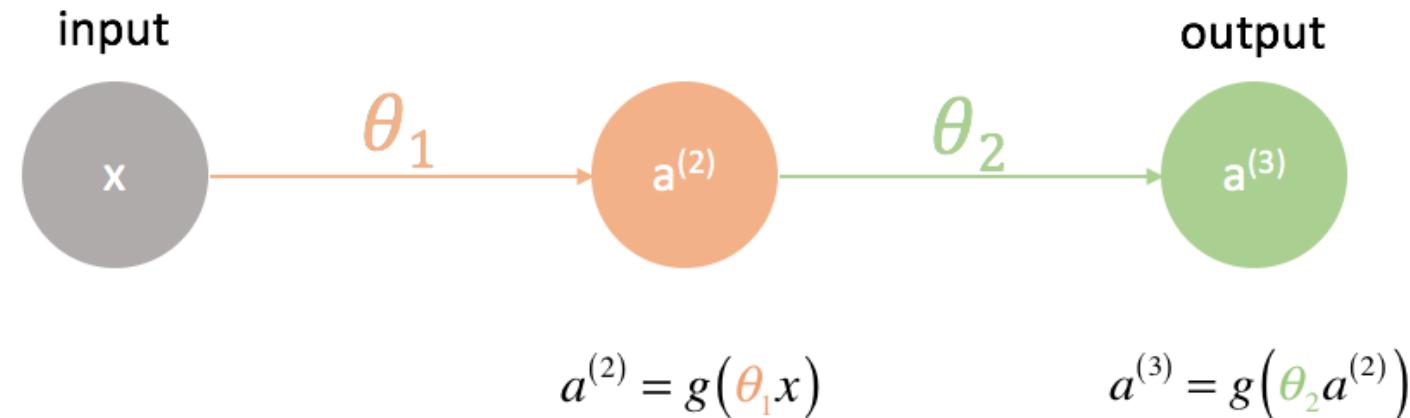
- Example of a 3-layer Deep Neural Network (DNN) – (input layer is not counted)



Courtesy: <http://cs231n.github.io/neural-networks-1/>

Essential Concepts: Back-propagation

- Back-propagation involves complicated mathematics.
 - Luckily, most DL Frameworks give you a one line implementation --
`model.backward()`

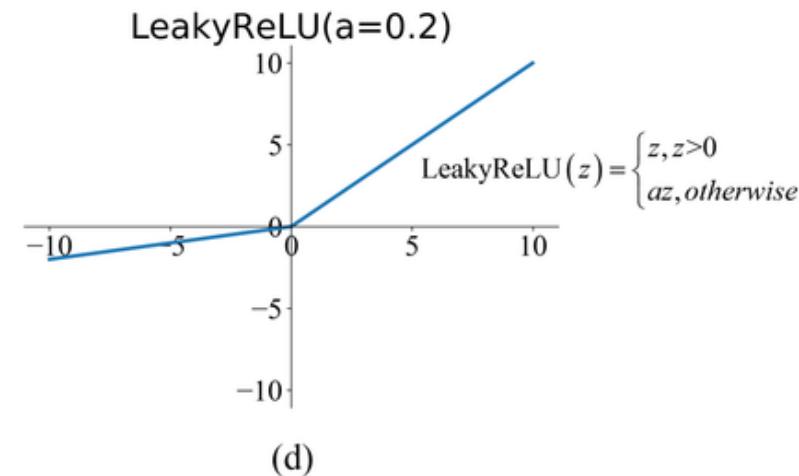
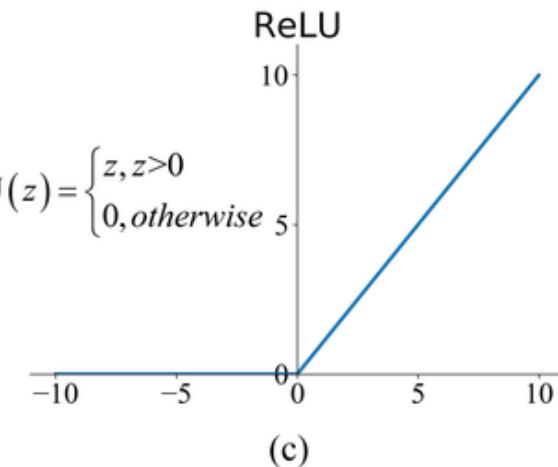
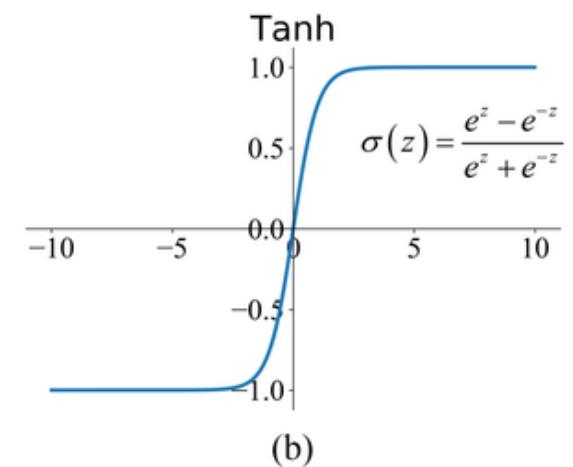
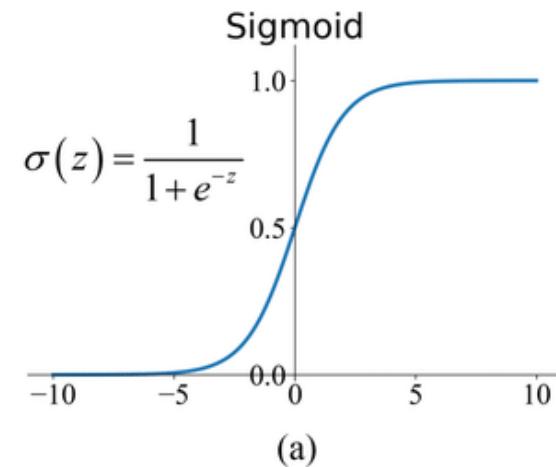
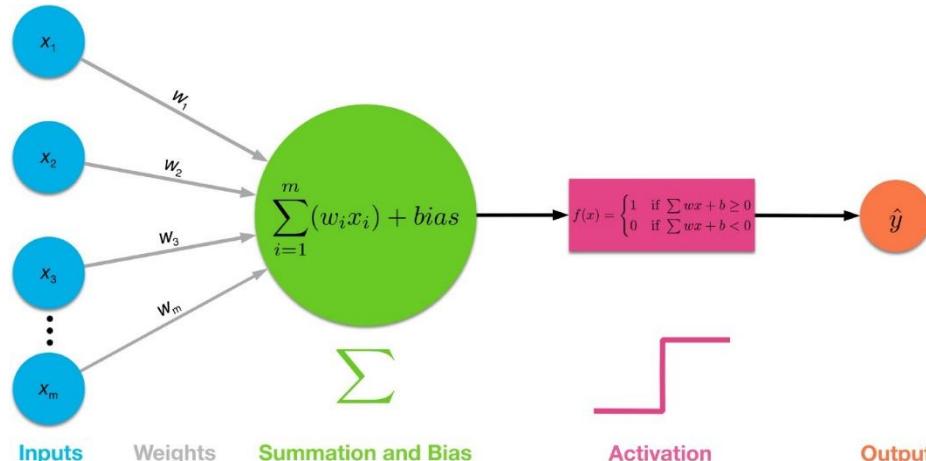


- What are Activation functions?
 - RELU (a Max fn.) is the most common activation fn.
 - Sigmoid, tanh, etc. are also used

Courtesy: <https://www.jeremyjordan.me/neural-networks-training/>

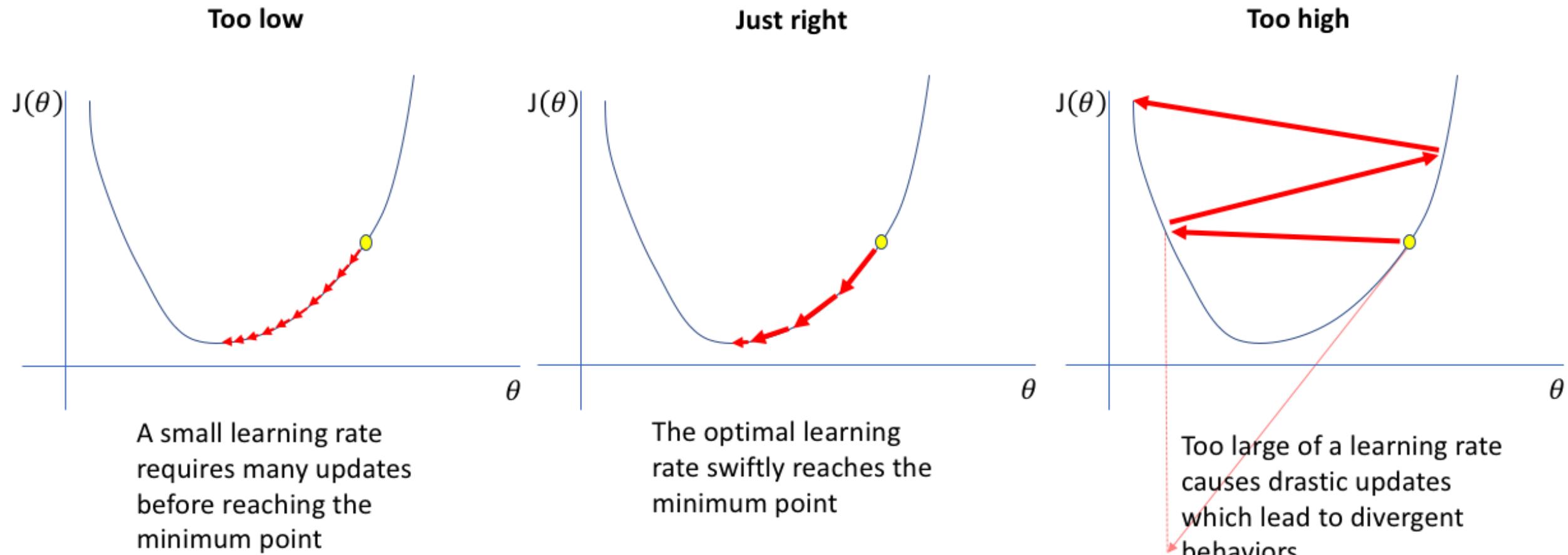
Essential Concepts: Activation Functions

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU



Courtesy: <https://journals.aps.org/pre/abstract/10.1103/PhysRevE.100.033308>

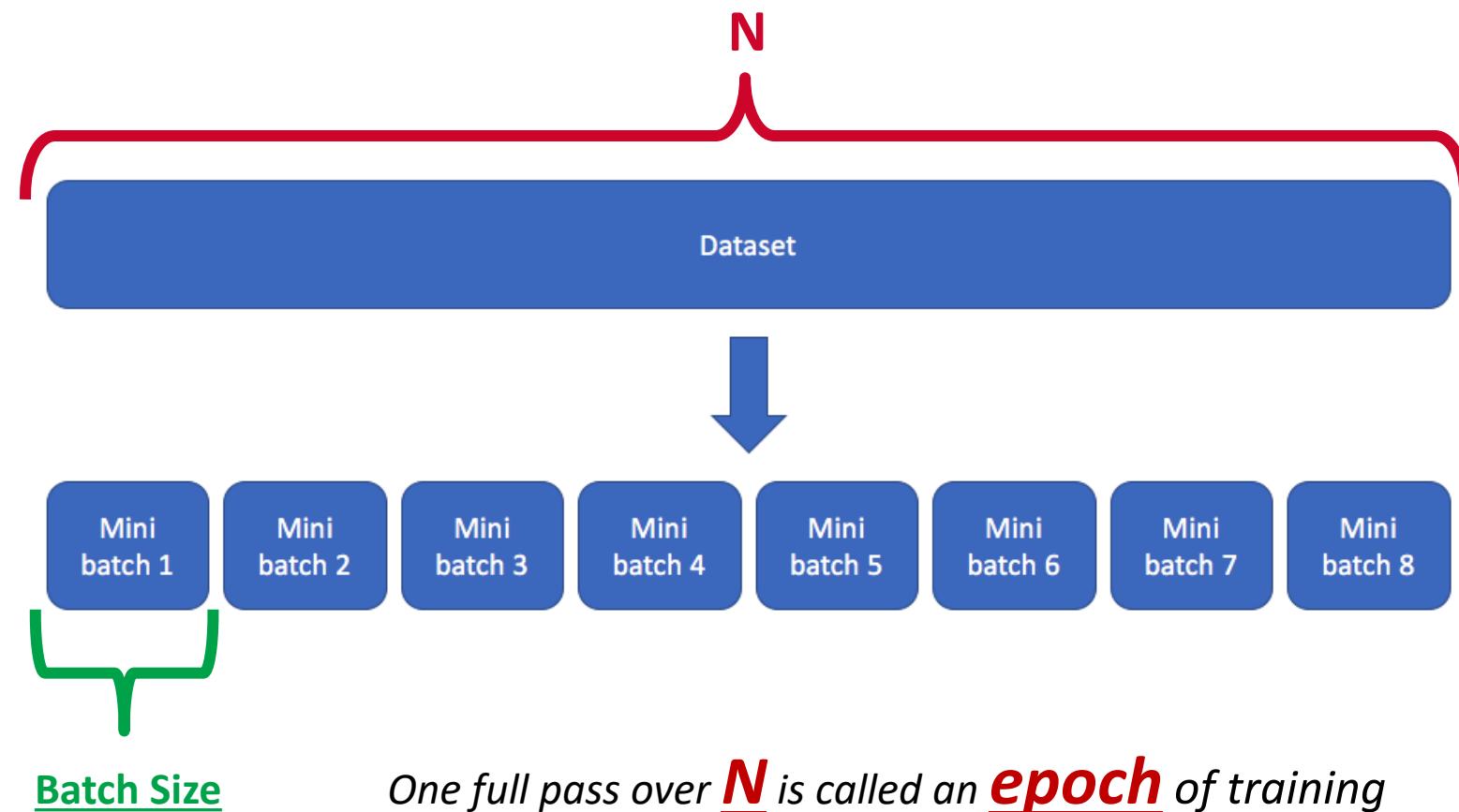
Essential Concepts: Learning Rate (α)



Courtesy: <https://www.jeremyjordan.me/nn-learning-rate/>

Essential Concepts: Batch Size

- Batched Gradient Descent
 - Batch Size = N
- Stochastic Gradient Descent
 - Batch Size = 1
- Mini-batch Gradient Descent
 - Somewhere in the middle
 - Common:
 - Batch Size = 64, 128, 256, etc.
- Finding the optimal batch size will yield the fastest learning.

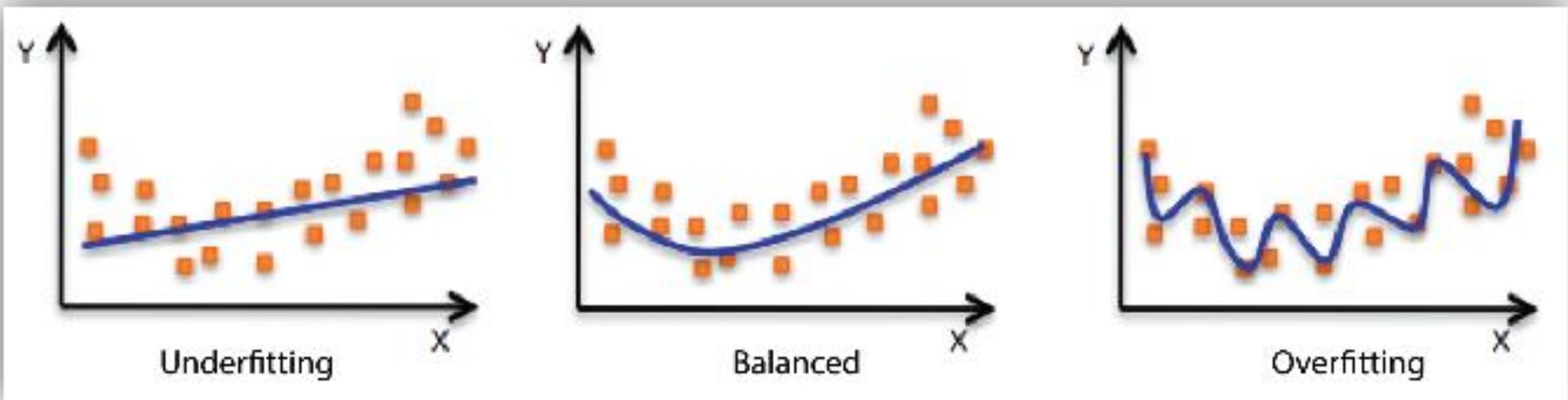


One full pass over N is called an epoch of training

Courtesy: <https://www.jeremyjordan.me/gradient-descent/>

Overfitting and Underfitting

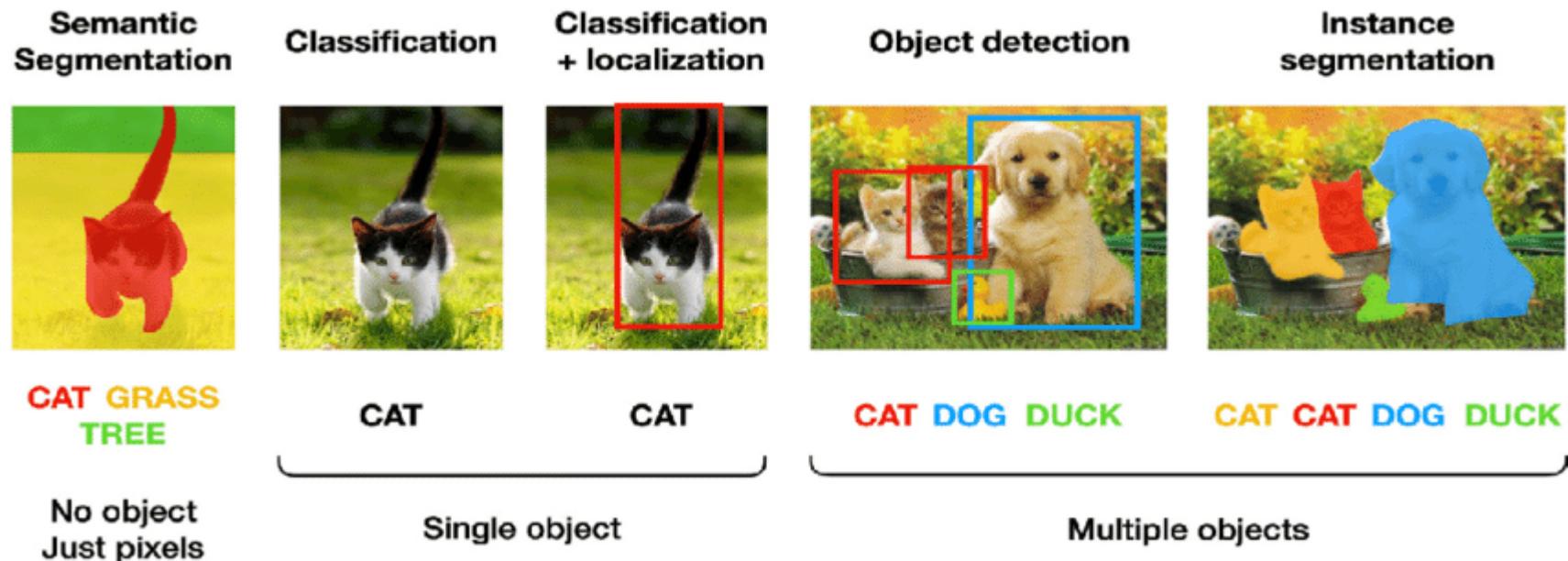
- Overfitting – **model > data** → so model is not learning but memorizing your data
- Underfitting – **data > model** → so model is not learning because it cannot capture the complexity of your data



Courtesy: <https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html>

What is Computer Vision (CV)?

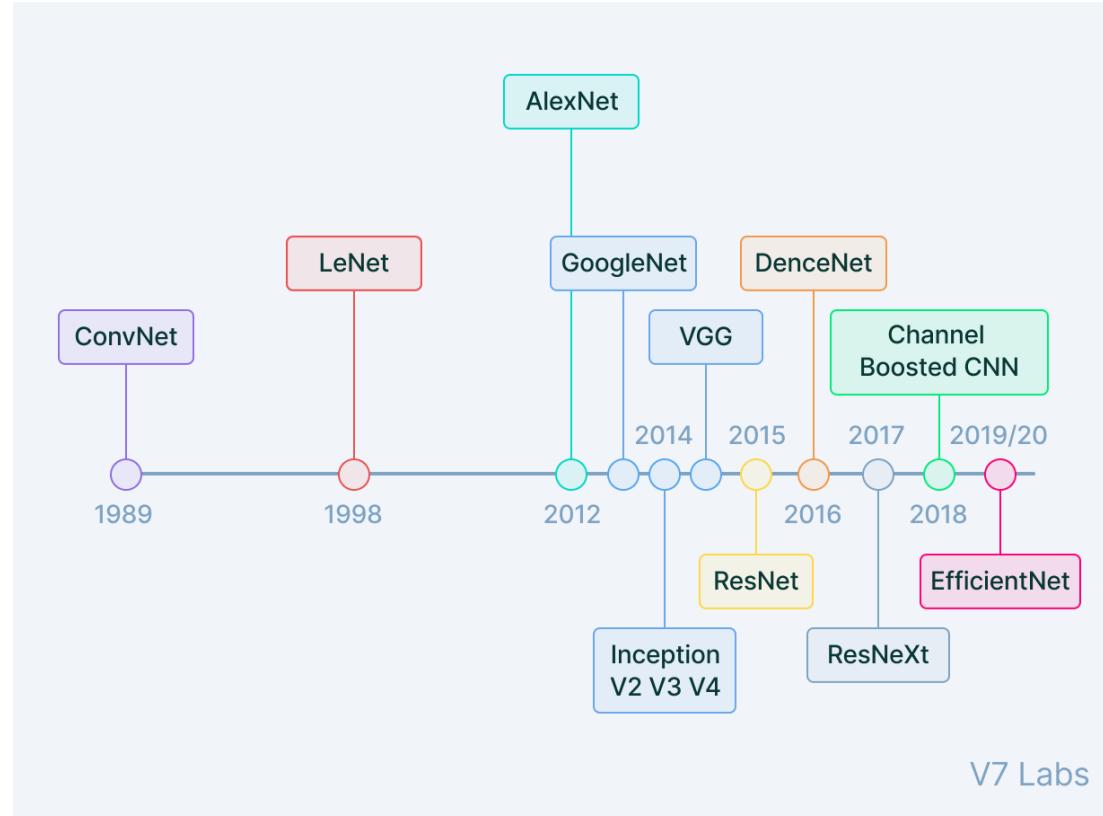
Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs — and take actions or make recommendations based on that information.



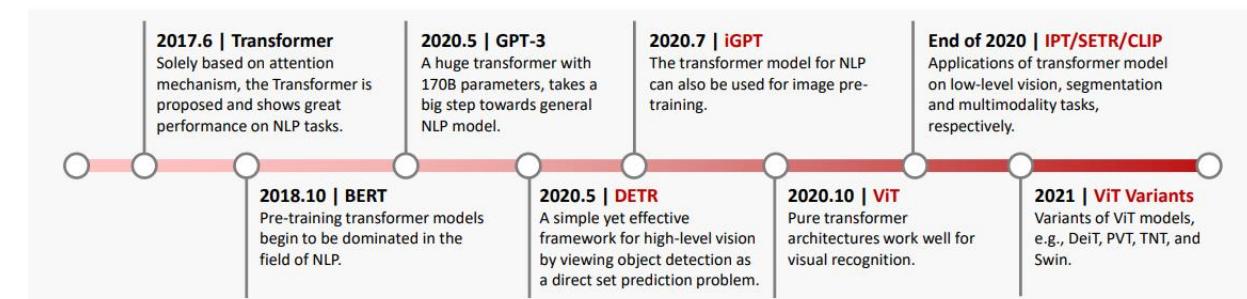
Courtesy: <https://www.ibm.com/topics/computer-vision>
<https://www.implantology.or.kr/articles/article/RvNO/>

Evolution of Computer Vision Models

CNN Architectures



Vision Transformer Architectures

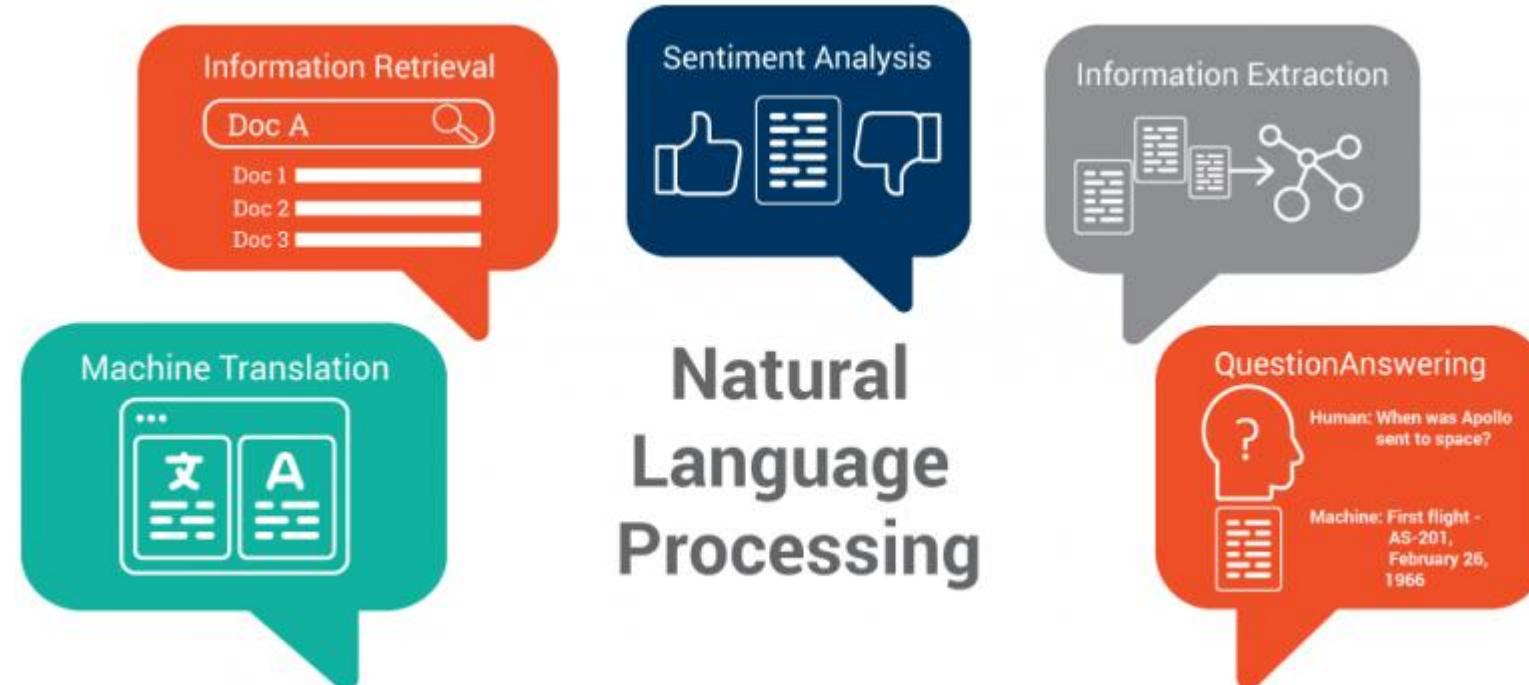


Courtesy: <https://www.v7labs.com/blog/convolutional-neural-networks-guide>

A Survey on Vision Transformer (Kai Han et. Al 2022) <https://arxiv.org/abs/2012.12556>

What is Natural Language Processing (NLP)?

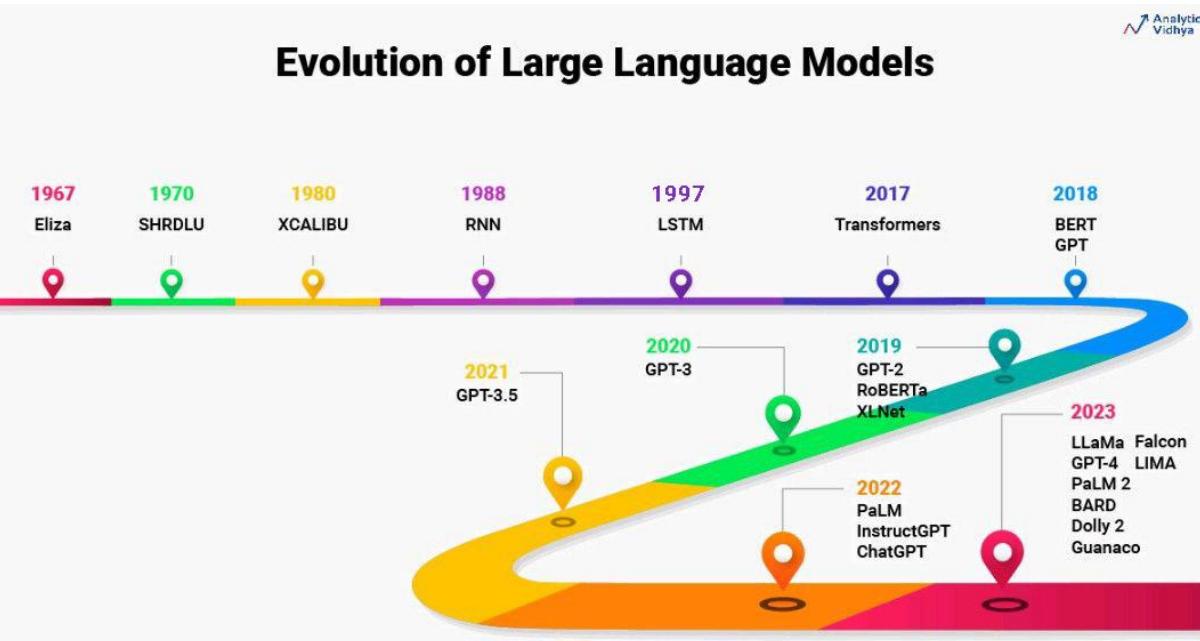
Natural Language Processing (NLP) is a subfield of artificial intelligence (AI) that focuses on the interaction between computers and human language. It aims to enable computers to understand, interpret, and generate human language in a valuable way.



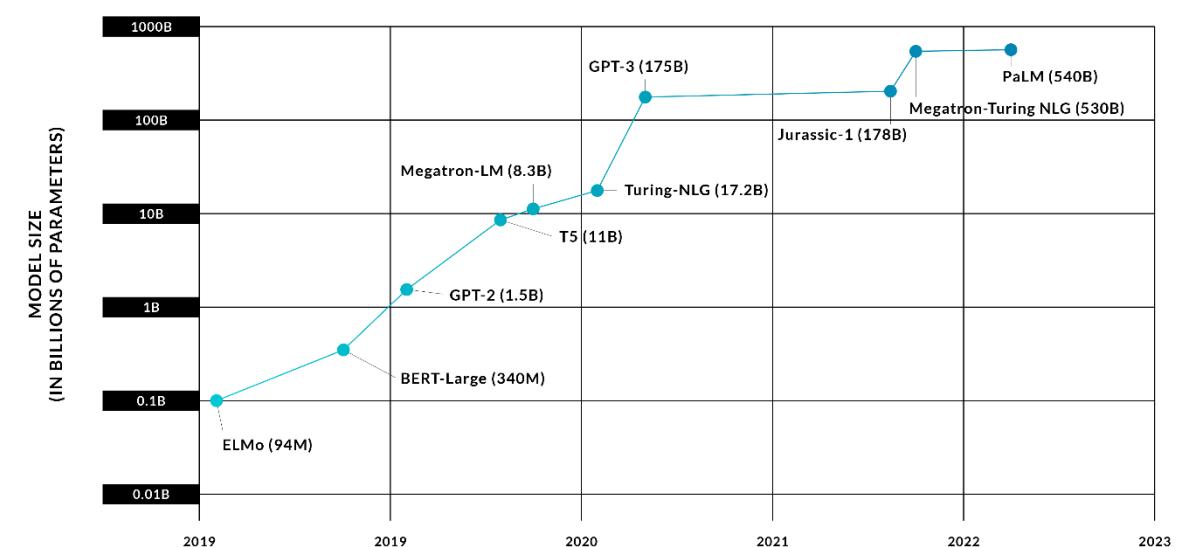
Courtesy: <https://www.ontotext.com/blog/top-5-semantic-technology-trends-2017/>

Evolution of Language Models

Evolution of Large Language Models



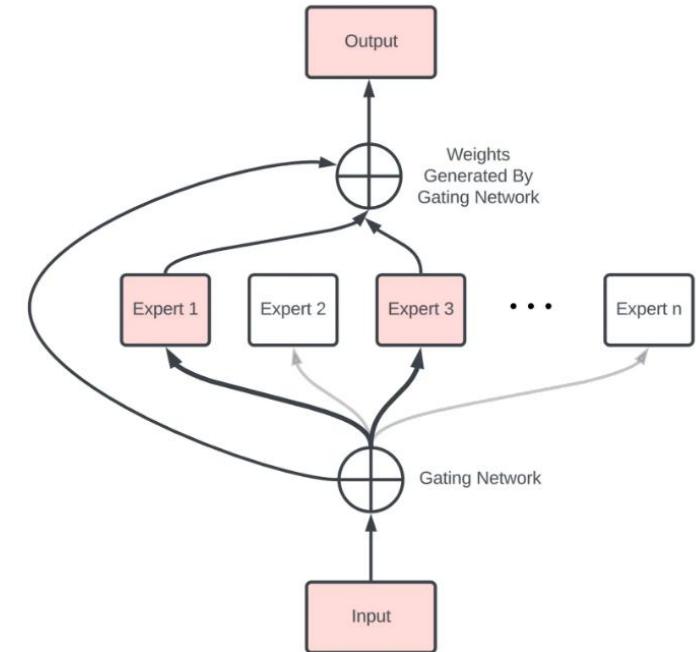
Language Model Sizes Over Time



Courtesy: <https://www.analyticsvidhya.com/blog/2023/07/build-your-own-large-language-models/>
<https://www.vinayiyengar.com/2022/08/04/the-promise-and-perils-of-large-language-models/>

Mixture of Experts (MoE)

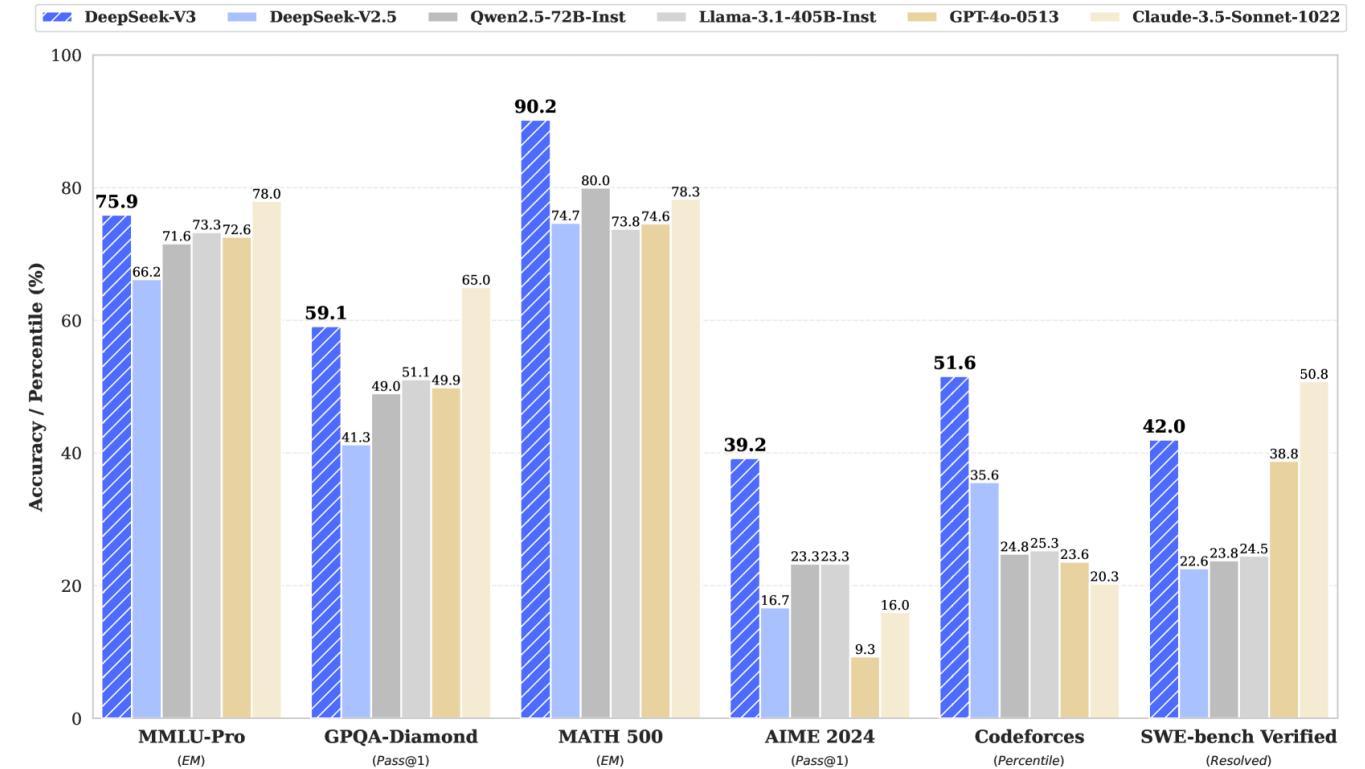
- Ensemble methods have long been powerful machine learning and deep learning methods to break down problems into easier subproblems
 - E.g. for vision classification, train separate “expert” models on subdomains (animal classifier, car classifier, etc), then route the incoming image to the appropriate model depending on its subclass (animal, car, etc)
 - E.g. for multilingual language modeling, train separate “expert” models for each language, then route incoming words to the appropriate model depending on its language
- Mixture-of-experts (MoE) models are ensembles of component “experts” coupled with a “gating” function that routes tokens to their appropriate expert
 - Model-type agnostic



Credit: <https://deepgram.com/learn/mixture-of-experts-ml-model-guide>

MoE State of the Art: DeepSeek-V3

- DeepSeek-V3 is a powerful Mixture-of-Experts (MoE) open-source model with 671B total parameters, 37B of which are active per token.
- It leverages Multi-head Latent Attention (MLA) and DeepSeekMoE architectures for efficient inference and cost-effective training.
- Designed for reasoning, DeepSeek excels in logic, pattern recognition, math, and tasks where typical generative AI models fall short.
- The training of DeepSeek-V3 is cost-effective due to the support of FP8 training and communication optimizations.



Benchmark performance of DeepSeek-V3 and its counterparts

Outline

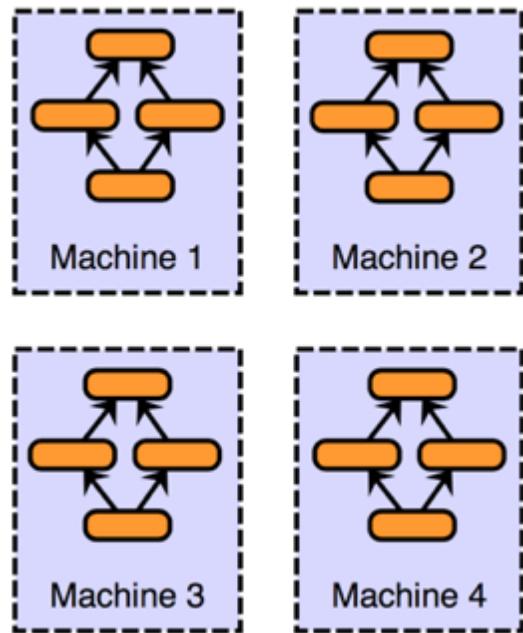
- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- **Distributed Data-Parallel Training**
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

The Need for Parallel and Distributed Training

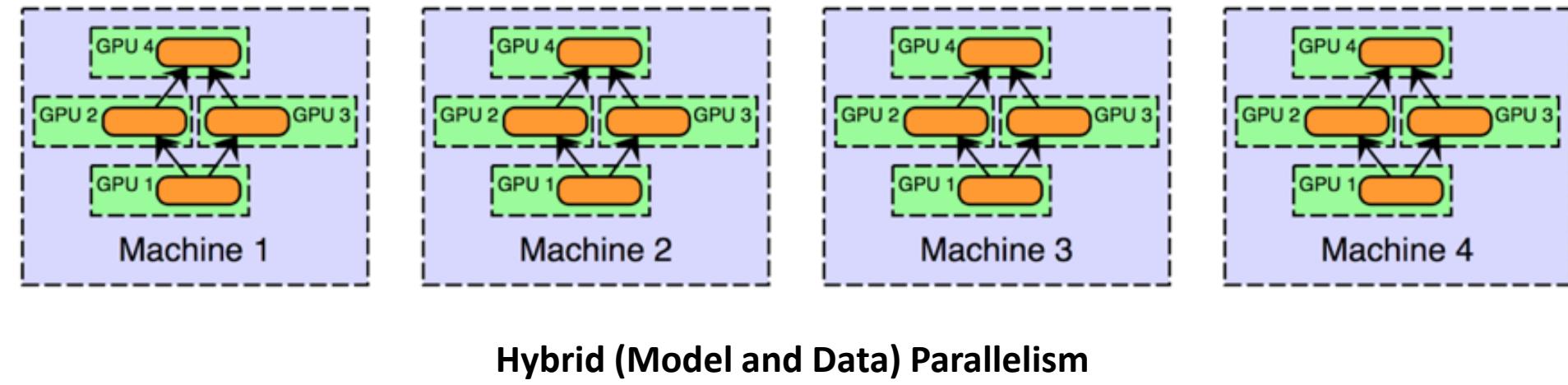
- Why do we need Parallel Training?
- Larger and Deeper models are being proposed
 - **Language Models:** RNNs -> Transformers -> BERT – GPT-(1,2,3,4)
 - **Vision Models:** AlexNet -> ResNet -> NASNet – AmoebaNet → Vision Transformers
 - DNNs require a lot of memory and a lot of computation
 - Larger models cannot fit a GPU's memory
- Single GPU training cannot keep up with ever-larger models
- Community has moved to multi-GPU training
- Multi-GPU in one node is good but there is a limit to Scale-up (8-16 GPUs)
- **Multi-node (Distributed or Parallel) Training is necessary!!**

Parallelization Strategies

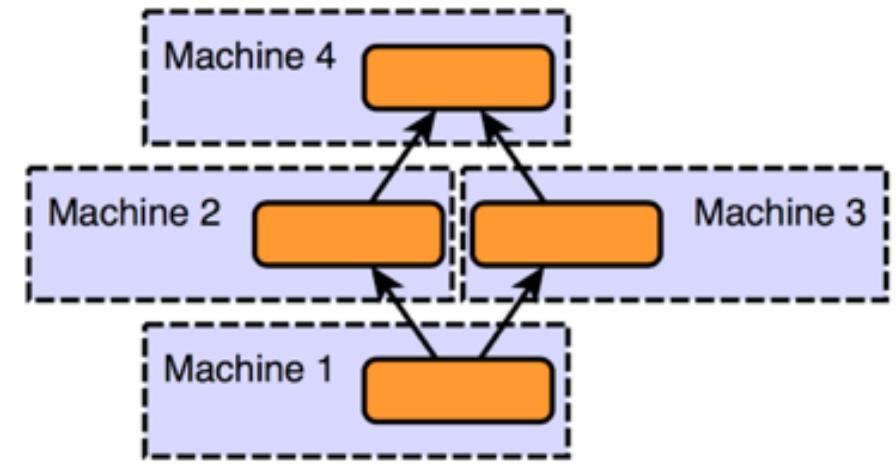
- Some parallelization strategies..
 - Data Parallelism or Model Parallelism
 - Hybrid Parallelism



Data Parallelism



Hybrid (Model and Data) Parallelism

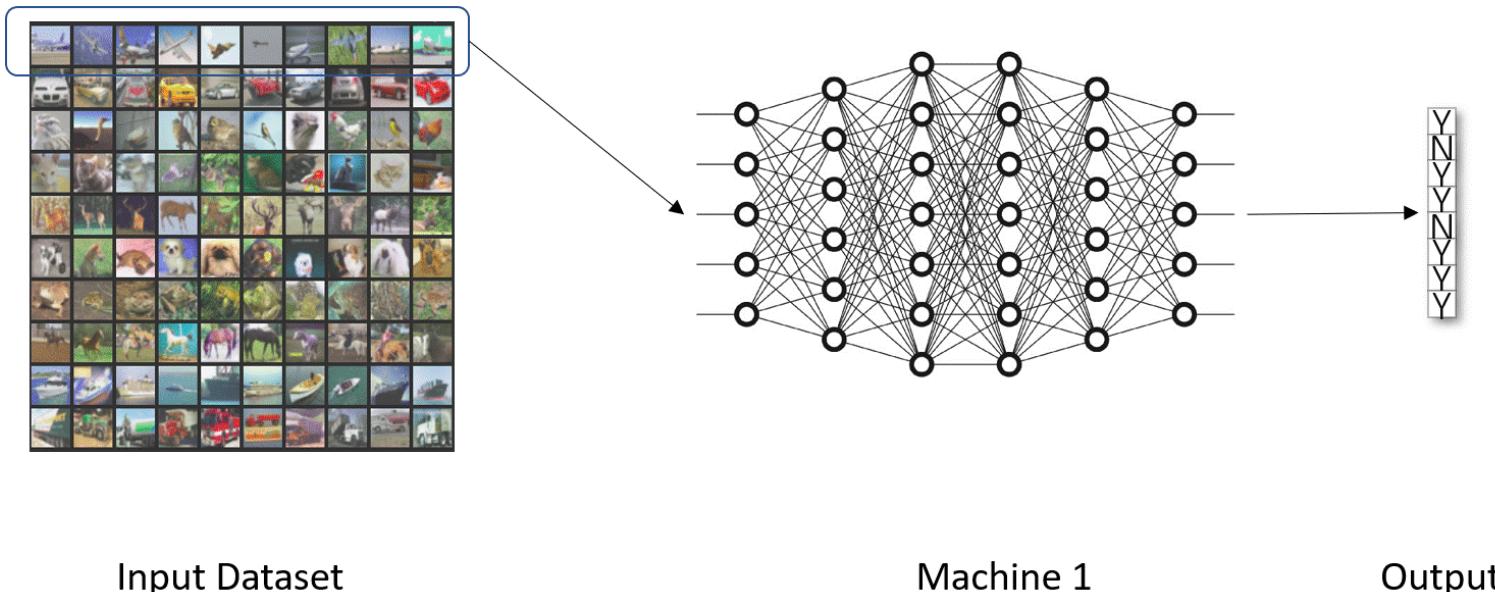


Model Parallelism

Courtesy: <http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks>

Need for Data Parallelism

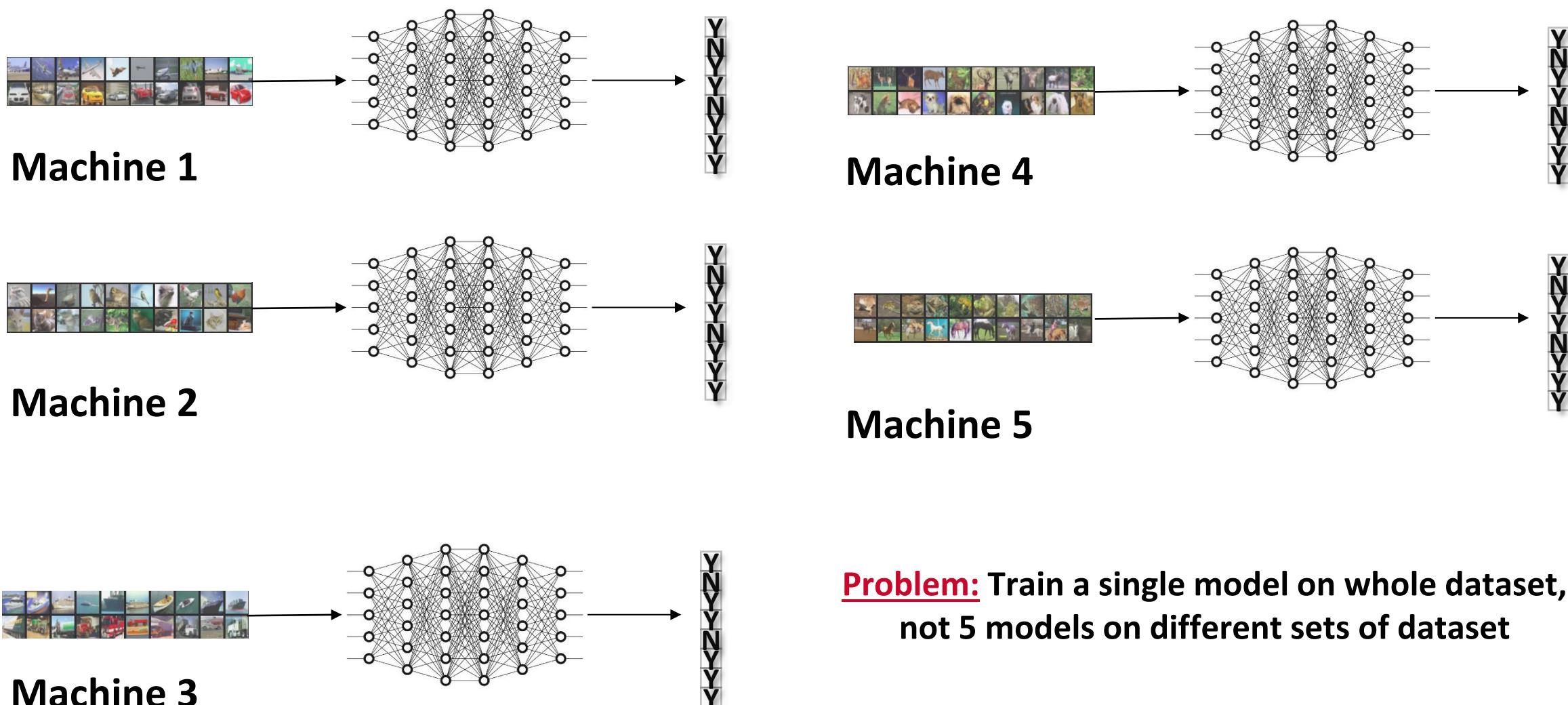
Mini-Batch Gradient Descent



Drawback: If the dataset has 1 million images, then it will take forever to train the model on such a big dataset

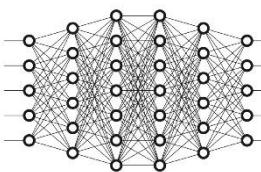
Solution: Can we use multiple machines to speedup the training of Deep learning models? (i.e. Utilize Supercomputers to Parallelize)

Need for Communication in Data Parallelism



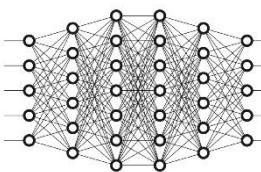
Data Parallelism

Machine 1



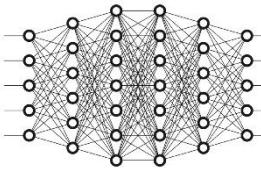
1	7	2
2	2	4
5	1	3

Machine 2



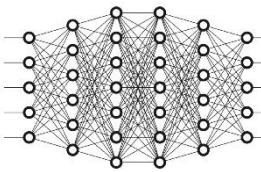
1	3	2
1	2	3
5	1	2

Machine 3



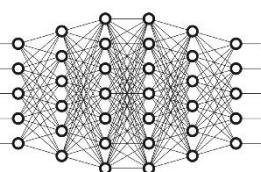
2	1	3
5	5	2
5	1	1

Machine 4



5	7	1
2	1	2
4	1	3

Machine 5



3	1	1
2	2	1
2	1	2

MPI
AllReduce

Gradients

12	19	9
12	12	12
21	5	11

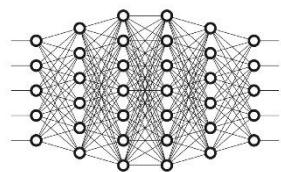
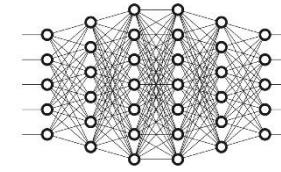
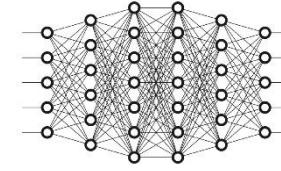
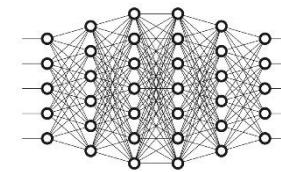
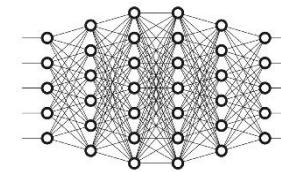
12	19	9
12	12	12
21	5	11

12	19	9
12	12	12
21	5	11

12	19	9
12	12	12
21	5	11

12	19	9
12	12	12
21	5	11

Reduced
Gradients



Allreduce Collective Communication Pattern

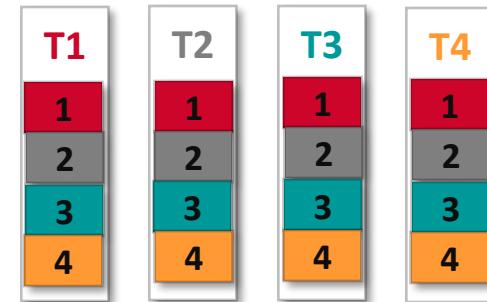
- Element-wise Sum data from all processes and sends to all processes

```
int MPI_Allreduce (const void *sendbuf, void * recvbuf, int count, MPI_Datatype datatype,  
MPI_Op operation, MPI_Comm comm)
```

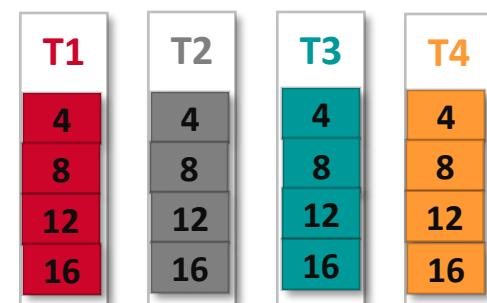
Input-only Parameters	
Parameter	Description
sendbuf	Starting address of send buffer
recvbuf	Starting address of recv buffer
type	Data type of buffer elements
count	Number of elements in the buffers
operation	Reduction operation to be performed (e.g. sum)
comm	Communicator handle

Input/Output Parameters	
Parameter	Description
recvbuf	Starting address of receive buffer

Sendbuf (Before)

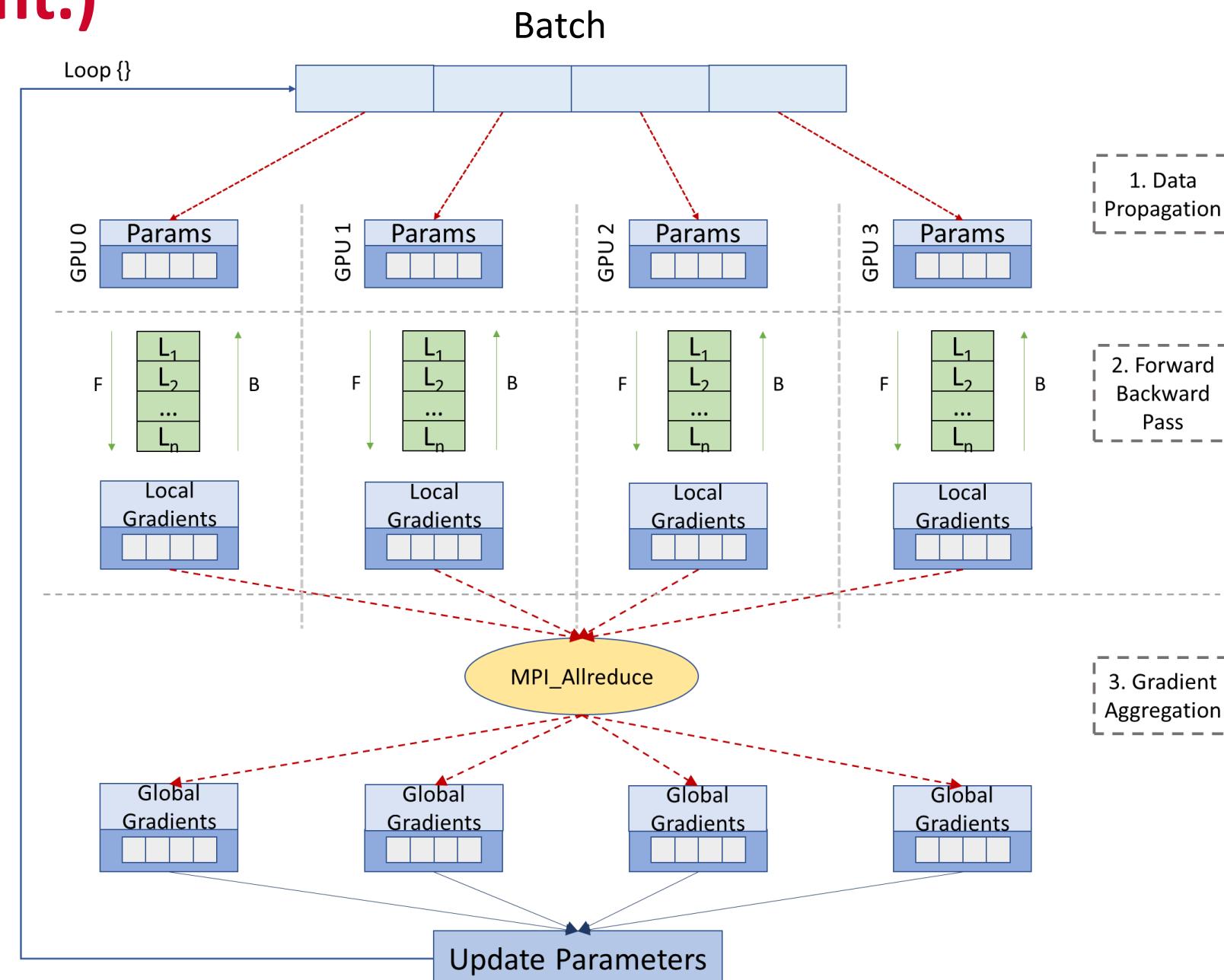


Recvbuf (After)

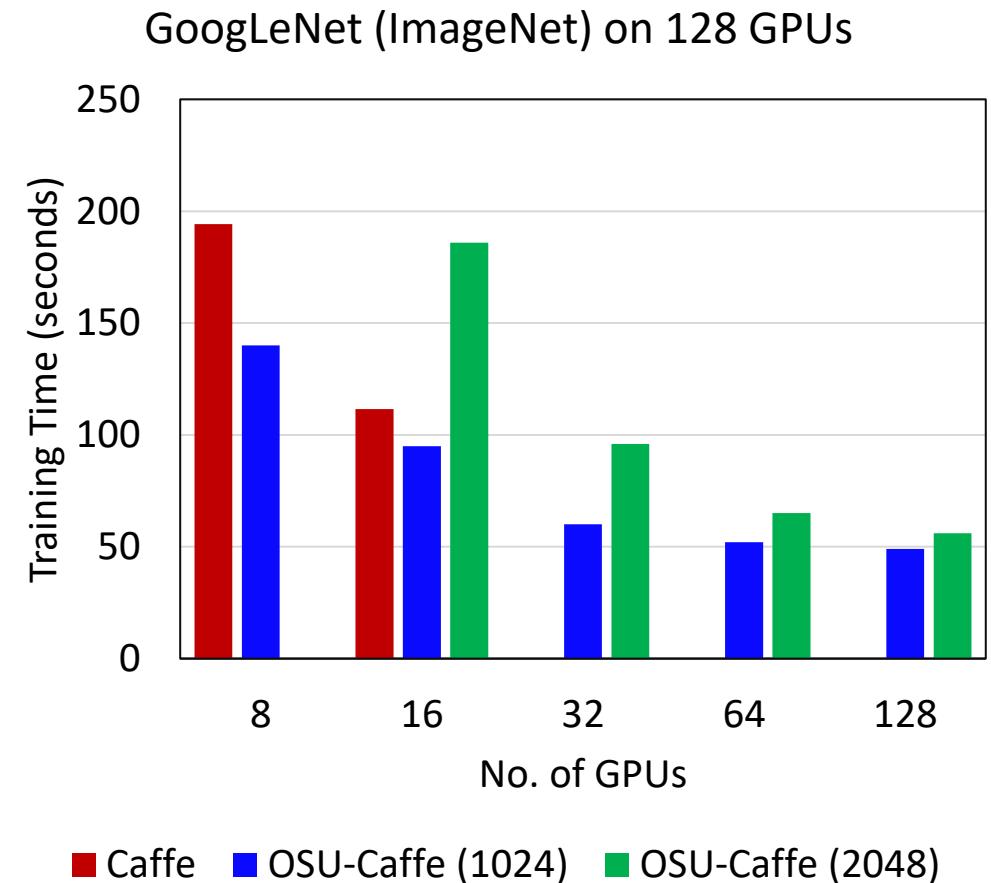
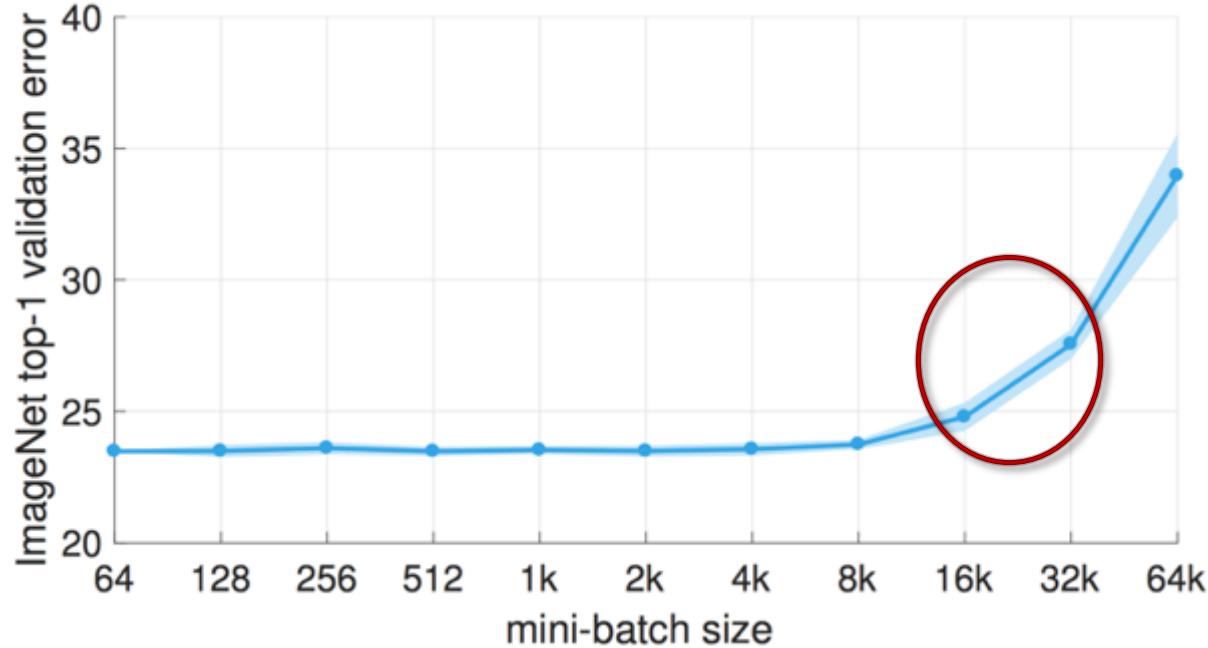


Data Parallelism (Cont.)

- **Step1:** Data Propagation
 - Distribute the Data among GPUs
- **Step2:** Forward Backward Pass
 - Perform forward pass and calculate the prediction
 - Calculate Error by comparing prediction with actual output
 - Perform backward pass and calculate gradients
- **Step3:** Gradient Aggregation
 - Call MPI_Allreduce to reduce the local gradients
 - Update parameters locally using global gradients



Impact of Large Batch Size



Large Batch Size is bad for Accuracy

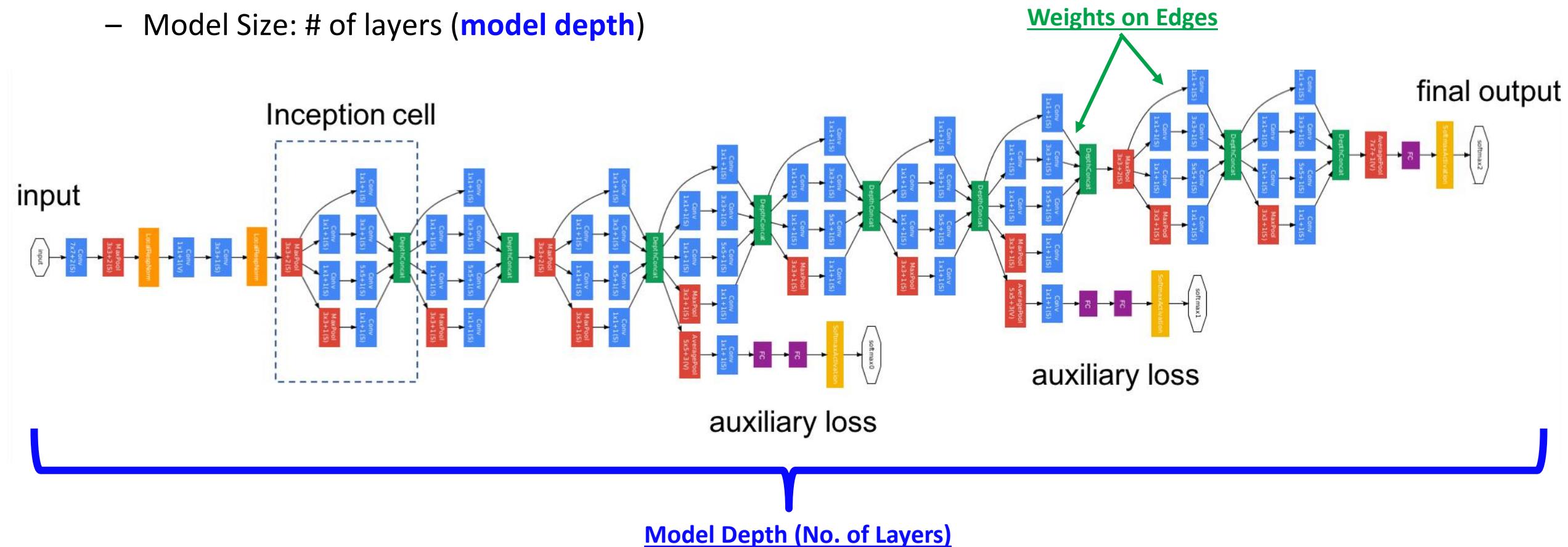
Courtesy: <https://research.fb.com/publications/imagenet1kin1h/>

But good for speed and scalability

A. A. Awan et al., S-Caffe: Co-designing MPI Runtimes and Caffe for Scalable Deep Learning on Modern GPU Clusters. PPoPP '17

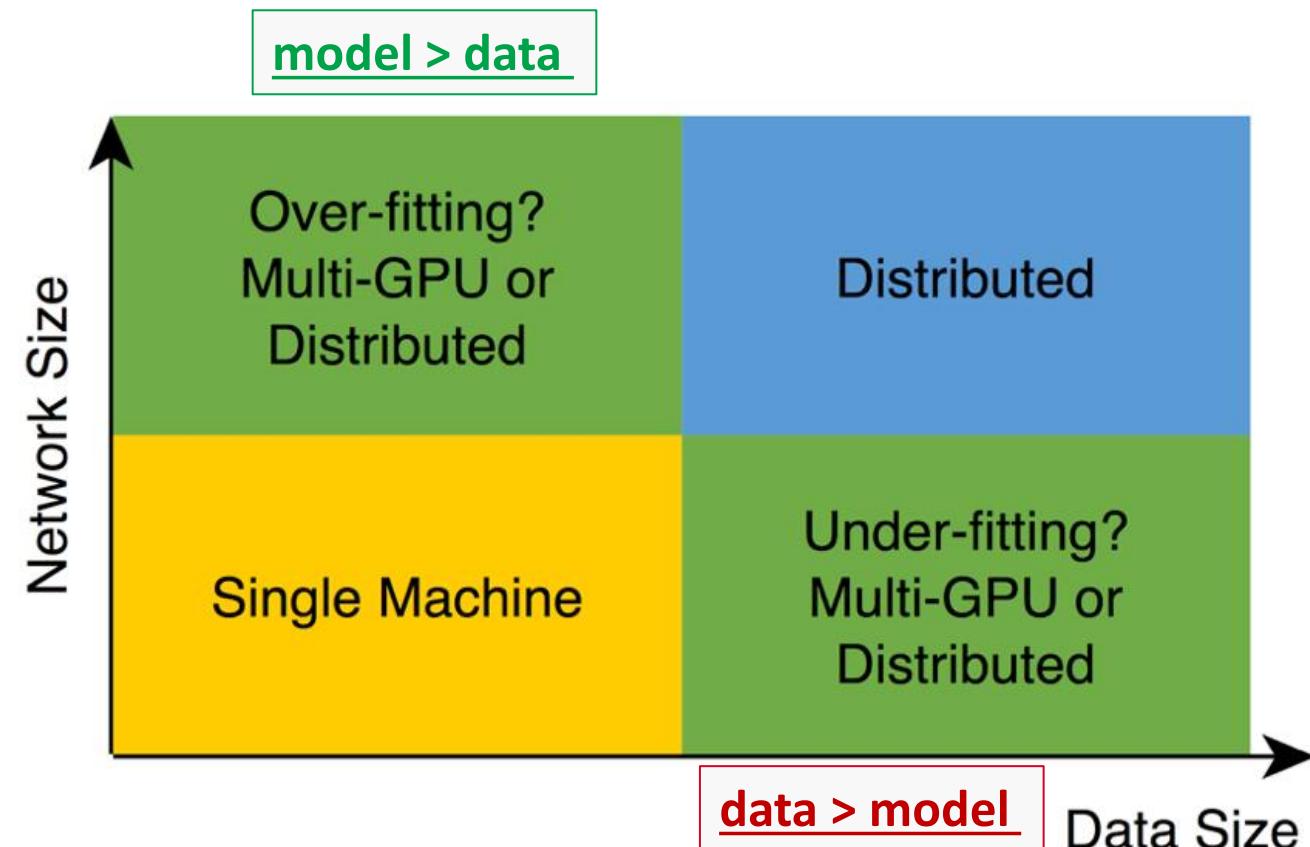
Essential Concepts: Model Size

- How to define the “size” of a model? (model is also called a DNN or a network)
- Size can mean several things and context is important
 - Model Size: # of parameters (**weights on edges**)
 - Model Size: # of layers (**model depth**)



Impact of Model Size and Dataset Size

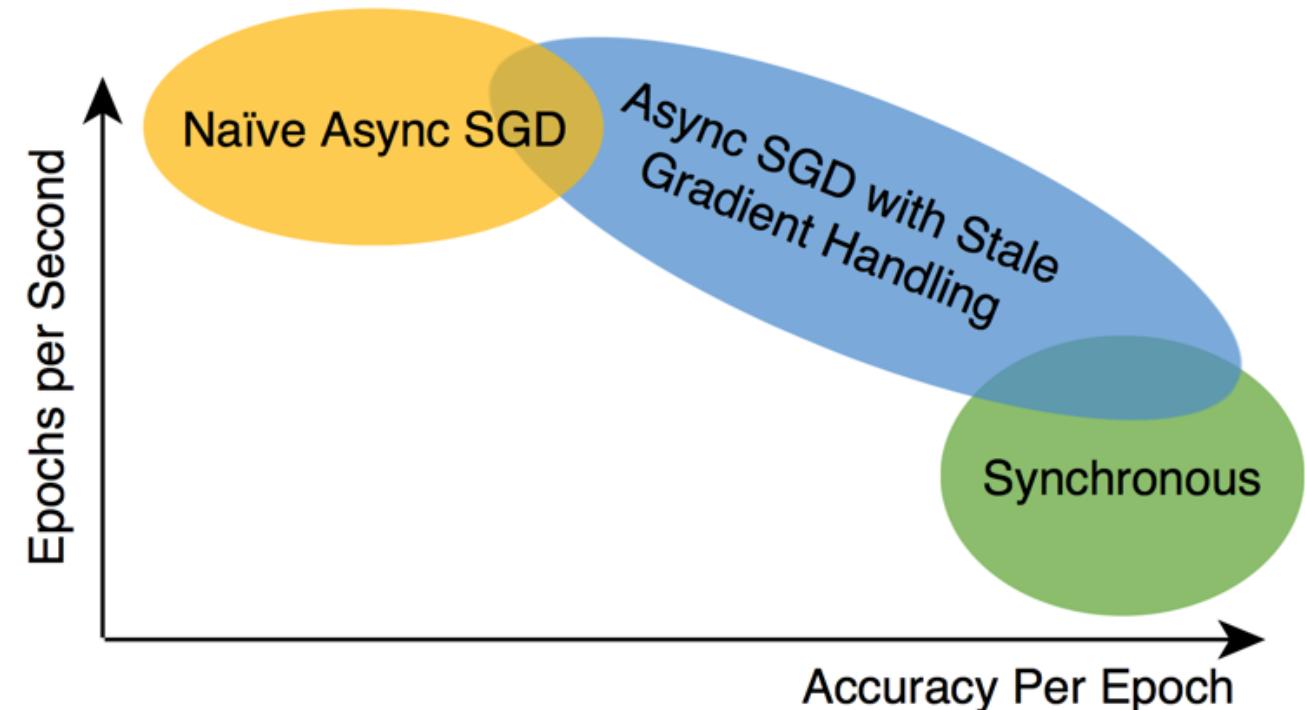
- *Large models → better accuracy*
- *More data → better accuracy*
- Single-node Training; good for
 - Small model and small dataset
- Distributed Training; good for:
 - Large models and large datasets



Courtesy: <http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks>

Synchronous vs. Asynchronous Training

- Epochs per second (EPS)?
 - A variant of images/second
 - Basically, what is the speed of training the model
- Accuracy per Epoch (APE)?
 - E.g. 60% in one full pass over the dataset
- Async → Higher EPS but lower APE
- Sync → Higher APE but lower EPS



Courtesy: <http://engineering.skymind.io/distributed-deep-learning-part-1-an-introduction-to-distributed-training-of-neural-networks>

Getting Set-up for the Hands-on Exercises

- You will run the experiments on the OSU RI2 cluster
- Please use the account name and password from <http://go.osu.edu/dltutorial>
- E.g. ssh ri2tut01@ri2.cse.ohio-state.edu and *tutorial01* as password
- Once on the shell, go to /opt/tutorials/dl-tutorial (copy/paste the following line)
`cd /opt/tutorials/hotihidl-tutorial`
- There is a folder for each lab (labs 1-2)
- Take a look at the README.md file for all scripts
 - copy/paste the run commands from README.md and not the slide deck

Lab 1 - DNN Training using PyTorch

- Objectives
 - How to train PyTorch and TensorFlow models on a single NVIDIA GPU?
 - How to perform distributed training of PyTorch and TensorFlow models on multiple GPUs using InfiniBand and NVIDIA GPUs?
- Tasks
 - Task 1: PyTorch Single GPU
 - Task 2: PyTorch Multi-GPU
 - Task 3: TensorFlow Single GPU
 - Task 4: TensorFlow Multi-GPU

Distributed Training with PyTorch using Horovod

- Examples to run data-parallel training with PyTorch using Horovod
- Available from: <https://github.com/horovod/horovod/tree/master/examples>
- To run ResNet50 with synthetic data with a single GPU, run

```
python pytorch_synthetic_benchmark.py \
--batch_size=32 \
----num-iters=10 \
```

Lab 1 – Task 1: Run PyTorch on a single GPU

```
$ cd /opt/tutorials/hot1-hidl-tutorial/lab1  
$ srun -N 1 --reservation=dltutorial run_pytorch_bench_single.sh  
+ python /opt/tutorials/hidl-env/horovod/examples/pytorch/pytorch_synthetic_benchmark.py --  
batch-size=64 --model=resnet50 --num-iters=5
```

```
.
```

```
Model: resnet50
```

V100

```
Batch size: 64
```

```
Number of GPUs: 1
```

```
Running warmup...
```

```
Running benchmark...
```

```
Iter #0: 336.0 img/sec per GPU
```

```
Iter #1: 336.0 img/sec per GPU
```

```
Iter #2: 336.0 img/sec per GPU
```

```
Iter #3: 336.0 img/sec per GPU
```

```
Iter #4: 335.8 img/sec per GPU
```

```
Img/sec per GPU: 336.0 +-0.2
```

```
Total img/sec on 1 GPU(s): 336.0 +-0.2
```

Lab 1 – Task 2: Run PyTorch on two nodes with 1 GPU/node (using MVAPICH-Plus)

```
$ srun -N 2 --reservation=dltutorial run_pytorch_bench_multi_mvp.sh
+ mpirun_rsh --export-all -np 2 --hostfile hosts_424919 python /opt/tutorials/hidl-
env/horovod/examples/pytorch/pytorch_synthetic_benchmark.py --batch-size=64 --model=resnet50 --
num-iters=5
.
.
.
Model: resnet50
Batch size: 64
Number of GPUs: 2
Running warmup...
Running benchmark...
Iter #0: 326.8 img/sec per GPU
Iter #1: 325.6 img/sec per GPU
Iter #2: 324.3 img/sec per GPU
Iter #3: 324.5 img/sec per GPU
Iter #4: 324.6 img/sec per GPU
Img/sec per GPU: 325.2 +-1.8
-----
Total img/sec on 2 GPU(s): 650.3 +-3.6
```

V100

~1.9X on
2 GPUs

Distributed Training with TensorFlow using Horovod

- Examples to run data-parallel training with TensorFlow using Horovod
- Available from: <https://github.com/horovod/horovod/tree/master/examples>
- To run ResNet50 with synthetic data with a single GPU, run

```
python tensorflow2_synthetic_benchmark.py\  
--batch_size=32 \  
----num-iters=10 \  
...
```

Lab 1 – Task 3: Run TensorFlow on a Single GPU

```
$ cd /opt/tutorials/sca-hidl-tutorial/lab1
$ srun -N 1 --reservation=dltutorial run_tf_bench_single.sh
+ python /opt/tutorials/hidl-env/horovod/examples/tensorflow2/tensorflow2_synthetic_benchmark.py -
-batch-size=64 --model=ResNet50 --num-iters=5
.
-----
.
Model: ResNet50
Batch size: 64
Number of GPUs: 1
V100
Running warmup...
Running benchmark...
Iter #0: 343.9 img/sec per GPU
Iter #1: 342.6 img/sec per GPU
Iter #2: 342.2 img/sec per GPU
Iter #3: 342.5 img/sec per GPU
Iter #4: 342.4 img/sec per GPU
Img/sec per GPU: 342.7 +-1.2
-----
Total img/sec on 1 GPU(s): 342.7 +-1.2
-----
```

Lab 1 – Task 4: Run TensorFlow on two nodes with 1 GPU/node (using MVAPICH3-GDR)

```
$ srun -N 2 --reservation=dltutorial run_tf_bench_multi_mvp.sh  
+ mpirun_rsh --export-all -np 2 --hostfile hosts_424948 python /opt/tutorials/hidl-  
env/horovod/examples/tensorflow2/tensorflow2_synthetic_benchmark.py --batch-size=64 --model=ResNet50 --num-iters=5  
.-----  
.-----  
Model: ResNet50  
Batch size: 64  
Number of GPUs: 2  
Running warmup...  
Running benchmark...  
Iter #0: 314.3 img/sec per GPU  
Iter #1: 314.2 img/sec per GPU  
Iter #2: 313.6 img/sec per GPU  
Iter #3: 314.3 img/sec per GPU  
Iter #4: 314.6 img/sec per GPU  
Img/sec per GPU: 314.2 +-0.6  
-----  
Total img/sec on 2 GPU(s): 628.4 +-1.3
```

V100

1.83X on
2 GPUs

Hands-on Exercises: Key Takeaways from DL labs

- Deep Learning models can be trained in multiple ways
 - Examples to run data-parallel training with Horovod are available at
[“https://github.com/horovod/horovod/tree/master/examples”](https://github.com/horovod/horovod/tree/master/examples)
 - Single/Multiple GPU jobs -- similar
 - Horovod can be configured MPI, GLOO, NCCL, and oneCCL.
 - MVAPICH-Plus with CUDA-aware design delivers near-linear speedup for multi-node training
 - User guide to install the full HiDL stack:
<https://hidl.cse.ohio-state.edu/userguide/horovod/>

Outline

- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- **Latest Trends in High-Performance Computing Architectures**
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

Drivers of Modern HPC Cluster Architectures



Multi-/Many-core
Processors



High Performance Interconnects -
InfiniBand
<1usec latency, 200Gbps Bandwidth>



Accelerators
high compute density, high
performance/watt
>1 TFlop DP on a chip



SSD, NVMe-SSD, NVRAM

- Multi-core/many-core technologies
- Remote Direct Memory Access (RDMA)-enabled networking (InfiniBand and RoCE)
- Solid State Drives (SSDs), Non-Volatile Random-Access Memory (NVRAM), NVMe-SSD
- Accelerators (NVIDIA GPGPUs)
- Available on HPC Clouds, e.g., Amazon EC2, NSF Chameleon, Microsoft Azure, etc.



Fugaku



Summit



Sierra



Sunway TaihuLight

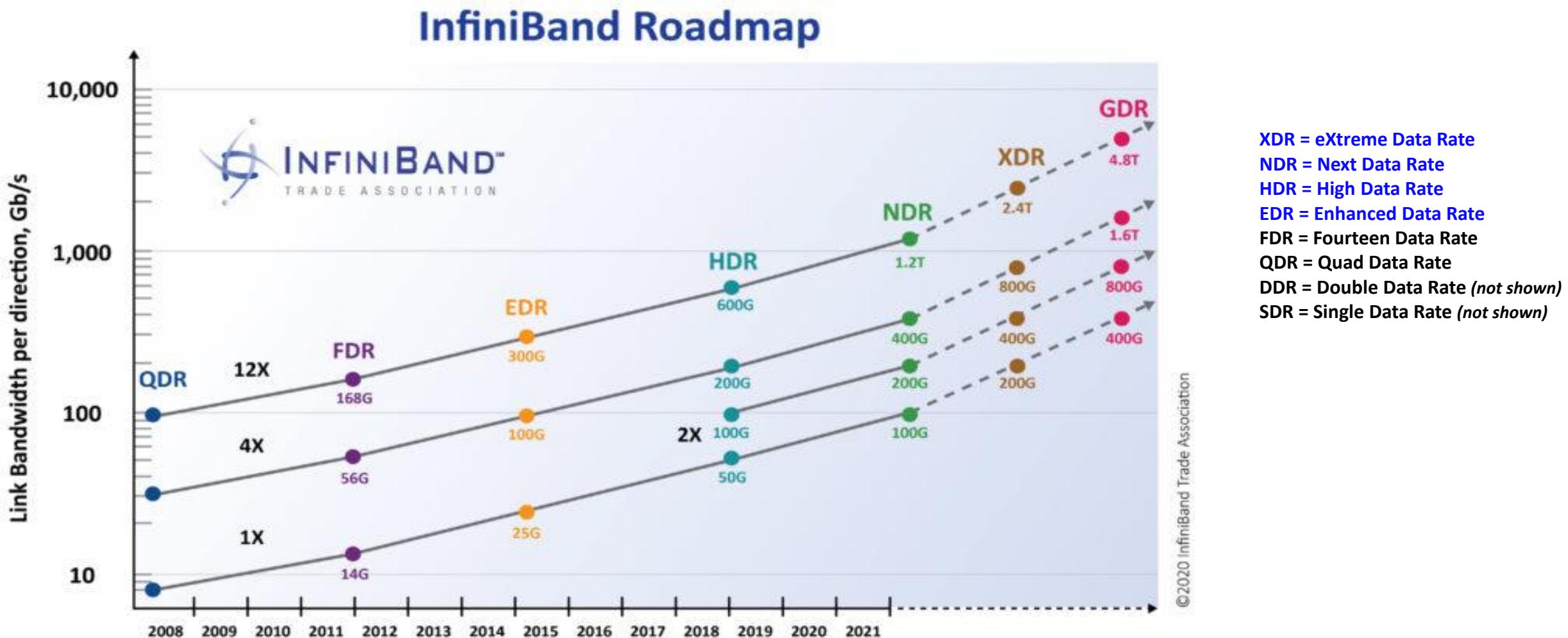
High-Performance Architectures for Distributed DL

- **Hardware Architectures**
 - Interconnects
 - InfiniBand, RoCE, Omni-Path, Slingshot, etc.
 - Processors
 - GPUs, Multi-/Many-core CPUs, Tensor Processing Unit (TPU), FPGAs, etc.
- Communication Middleware
 - Message Passing Interface (MPI)
 - CUDA-Aware MPI
 - NVIDIA NCCL

Overview of High Performance Interconnects

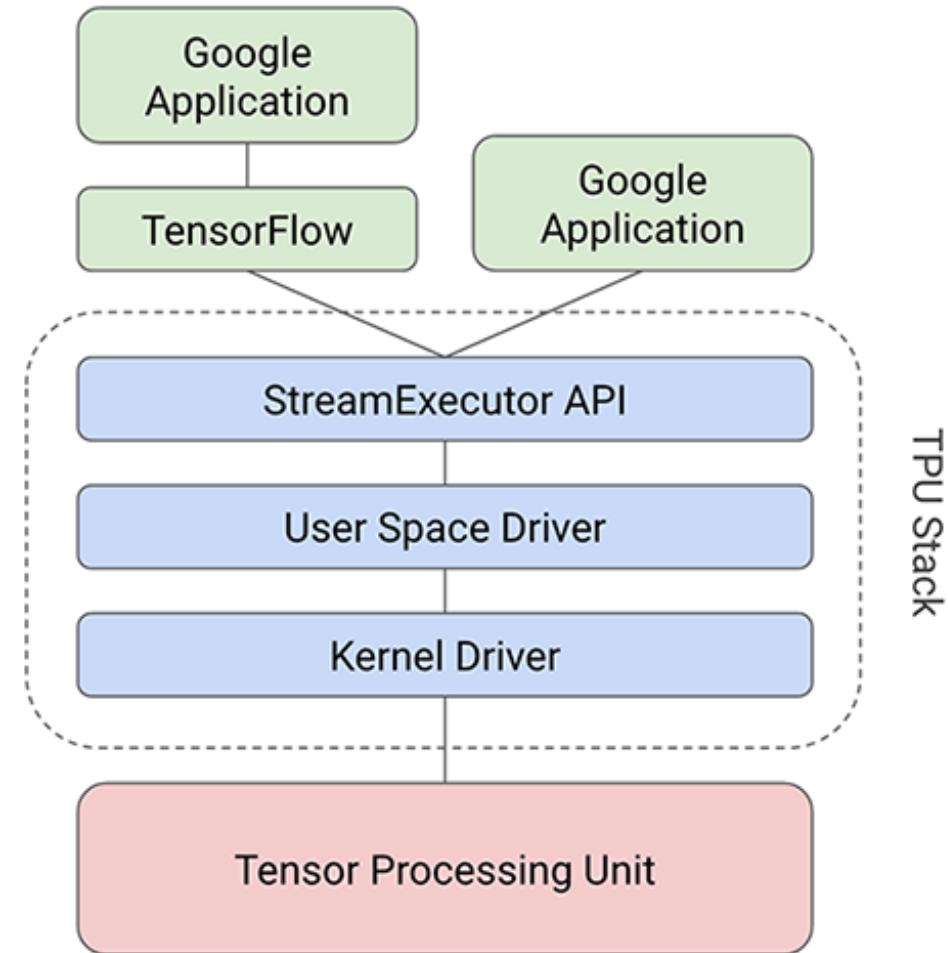
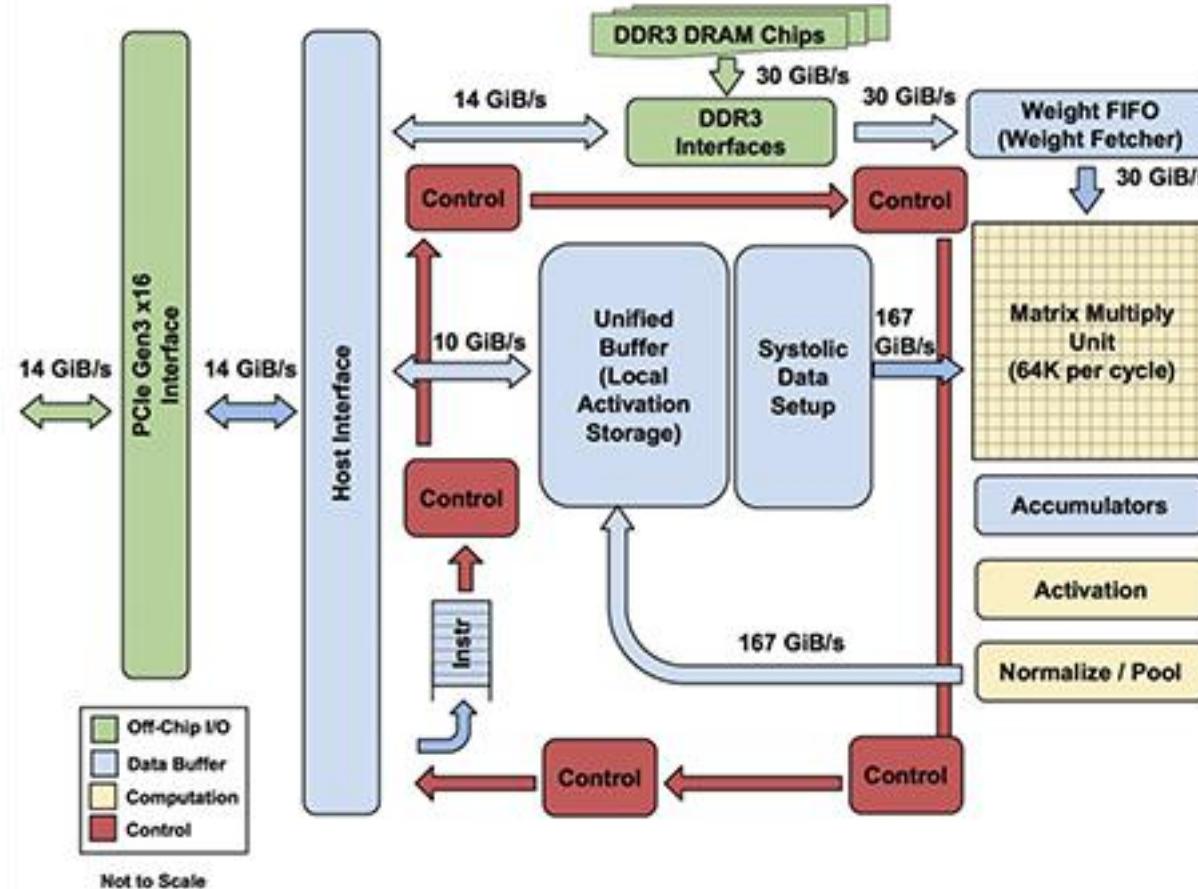
- High-Performance Computing (HPC) has adopted advanced interconnects and protocols
 - InfiniBand (IB)
 - Omni-Path
 - High Speed Ethernet 10/25/40/50/100/200 Gigabit Ethernet/iWARP
 - RDMA over Converged Enhanced Ethernet (RoCE)
- Very Good Performance
 - Low latency (few micro seconds)
 - High Bandwidth (400 Gb/s with NDR InfiniBand)
 - Low CPU overhead (5-10%)
- OpenFabrics software stack with IB, Omni-Path, iWARP and RoCE interfaces are driving HPC systems
- Many such systems in Top500 list

InfiniBand Link Speed Standardization Roadmap



Courtesy: InfiniBand Trade Association

Hardware for DNN Training and Inference: TPUs



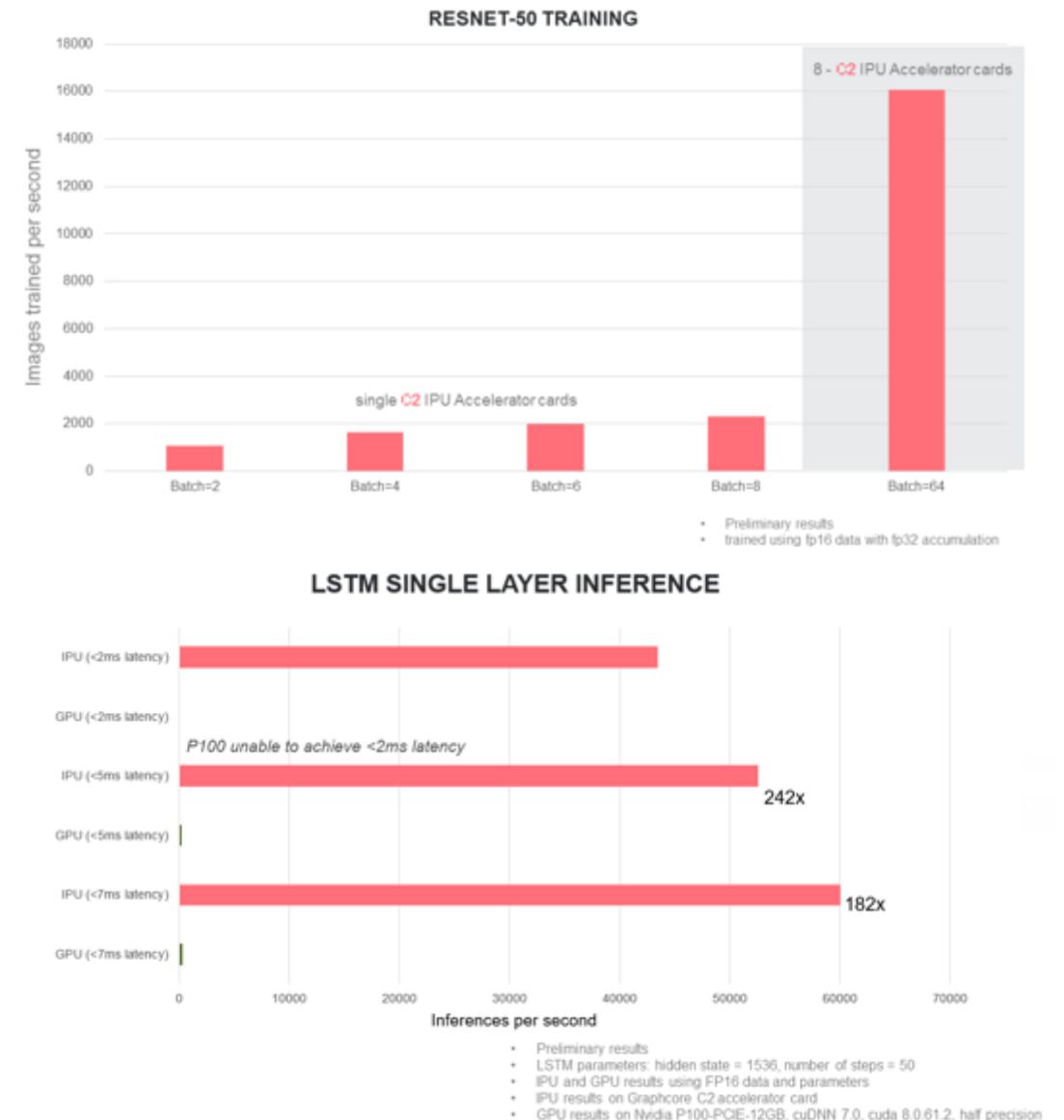
- CISC style instruction set
- Uses systolic arrays as the heart of multiply unit

Courtesy: <https://cloud.google.com/blog/big-data/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-tpu>

: <https://www.nextplatform.com/2017/04/05/first-depth-look-googles-tpu-architecture/>

Hardware for DNN Training and Inference: IPUs

- Specifically designed for AI workloads – an Intelligence Processing Unit (IPU)
 - Massively parallel
 - Low-precision floating-point compute
 - Higher compute density
- Early benchmarks show 10-100x speedup over GPUs
 - Presented at NIPS 2017
- HPC Wire: Microsoft Azure IPU instances
<https://www.hpcwire.com/2019/11/15/microsoft-azure-adds-graphcores-ipu/>



Courtesy: <https://www.graphcore.ai/posts/preliminary-ipu-benchmarks-providing-previously-unseen-performance-for-a-range-of-machine-learning-applications>

Hardware for DNN Training: Habana Gaudi

- Habana Labs – Training Accelerator called Gaudi – (HotChips ‘19)
- Gaudi – AI processor with RoCE integrated
- Gaudi software – Enables high-level frameworks
- ***Intel has acquired Habana for \$2 billion!***

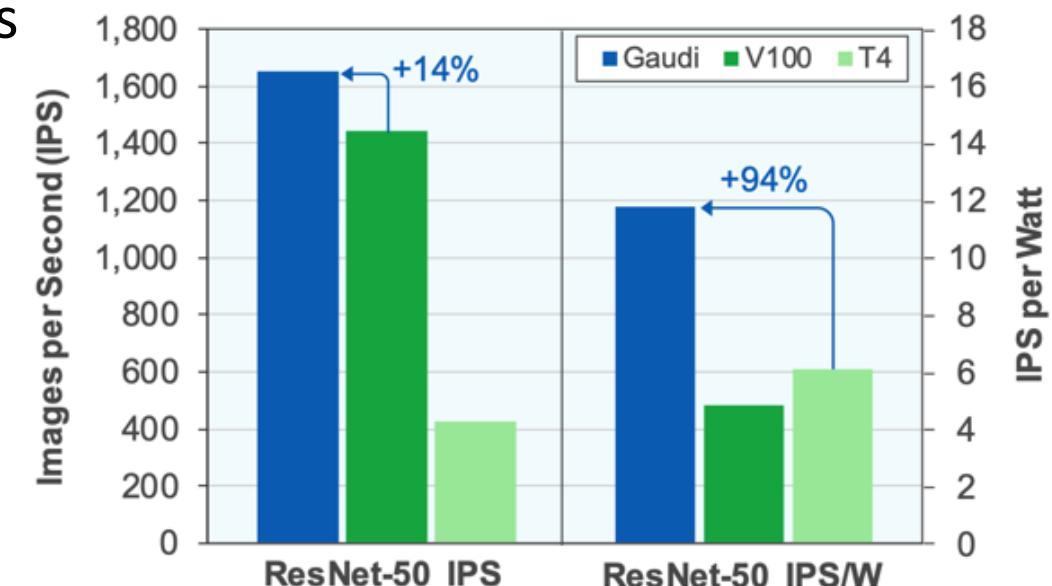
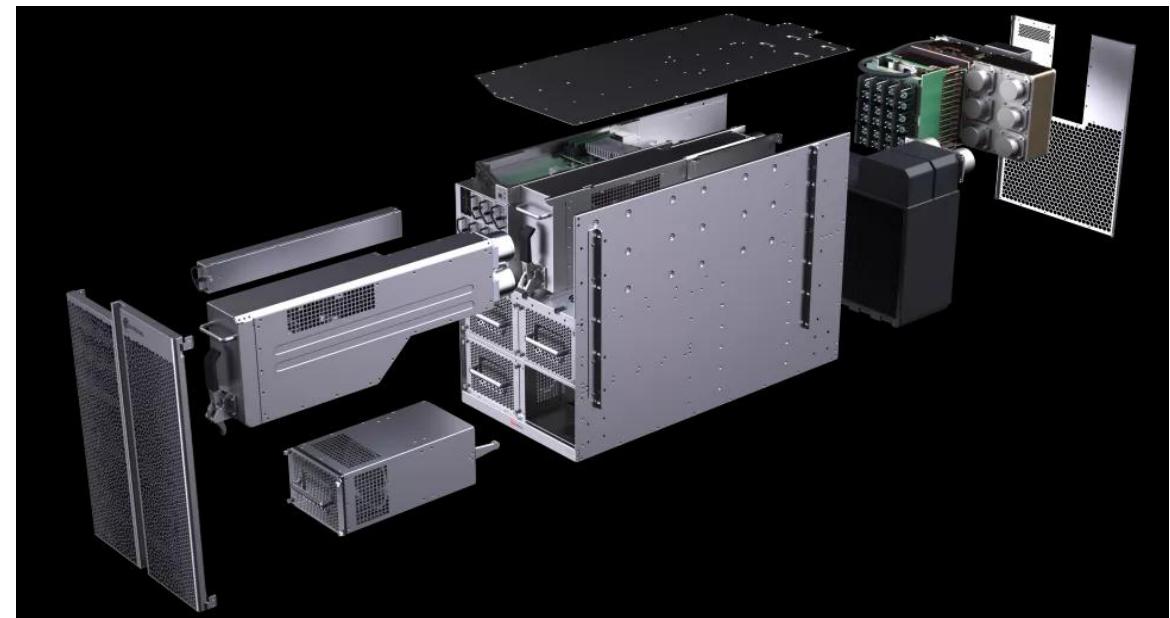
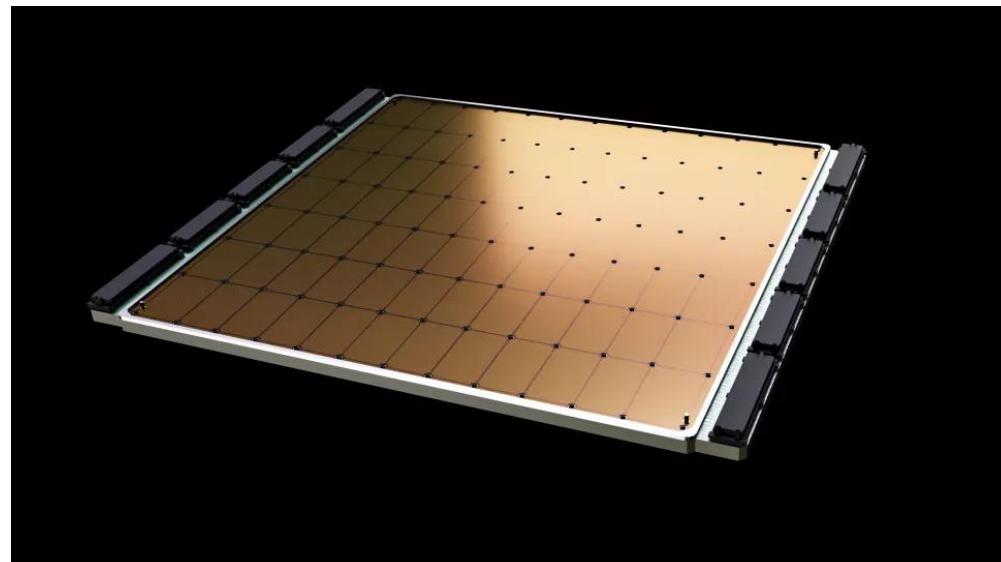


Figure 1. Gaudi emulated performance. For training the simple ResNet-50 model, Habana's Gaudi card offers throughput similar to that of Nvidia's high-end V100 GPU at half the power. It also beats Nvidia's Tesla T4 card in performance per watt.

Courtesy: <https://habana.ai/wp-content/uploads/2019/06/Habana-Offers-Gaudi-for-AI-Training.pdf>

Hardware for DNN Training: Cerebras

- Cerebras: First-Gen Wafer-Scale Engine (WSE) contains 400,000 Sparse Linear Algebra Compute (SLAC) Cores
- Swarm Communication fabric in a 2D mesh with 100 Pb/s of bandwidth
- Teased World's Largest Chip with **2.6 Trillion** 7nm Transistors and 850000 Cores (HotChips '20)

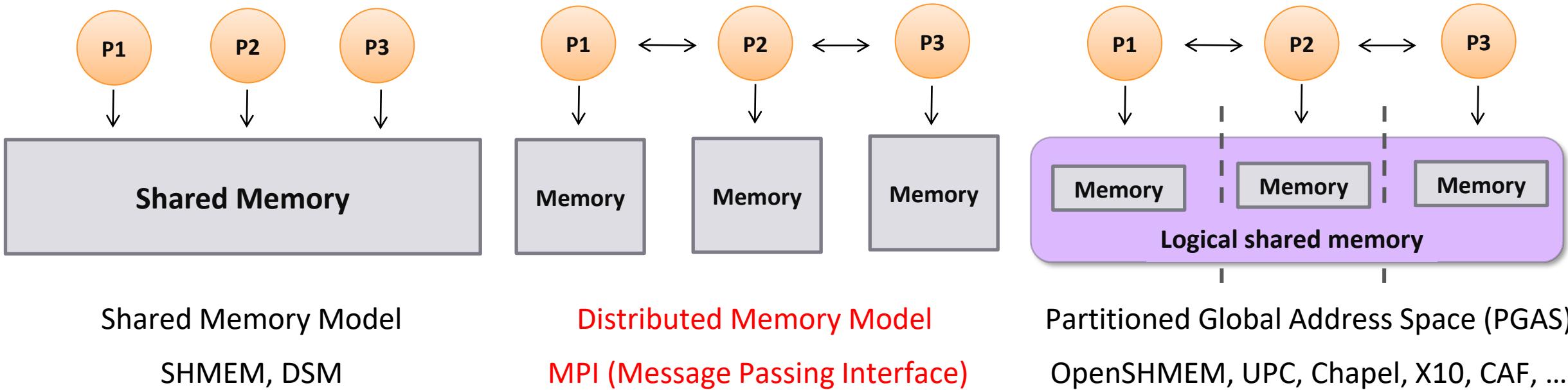


Courtesy: <https://www.cerebras.net/product/#chip>, <https://www.tomshardware.com/news/worlds-biggest-chip-cerebras-7nm-26-trillion-transistors-850000-cores-wafer-scale-engine>

High-Performance Architectures for Distributed DL

- Hardware Architectures
 - Interconnects
 - InfiniBand, RoCE, Omni-Path, etc.
 - Processors
 - GPUs, Multi-/Many-core CPUs, Tensor Processing Unit (TPU), FPGAs, etc.
- **Communication Middleware**
 - **Message Passing Interface (MPI)**
 - CUDA-Aware MPI
 - **NVIDIA NCCL**

Parallel Programming Models Overview



- Programming models provide abstract machine models
- Models can be mapped on different types of systems
 - e.g. Distributed Shared Memory (DSM), MPI within a node, etc.
- PGAS models and Hybrid MPI+PGAS models are gradually receiving importance

Allreduce Collective Communication Pattern

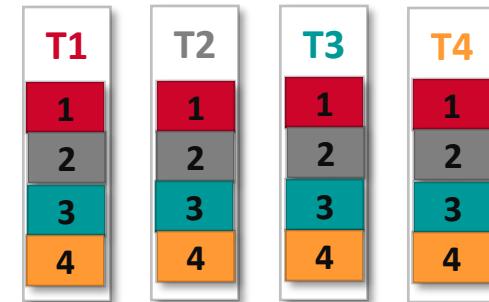
- Element-wise Sum data from all processes and sends to all processes

```
int MPI_Allreduce (const void *sendbuf, void * recvbuf, int count, MPI_Datatype datatype,  
MPI_Op operation, MPI_Comm comm)
```

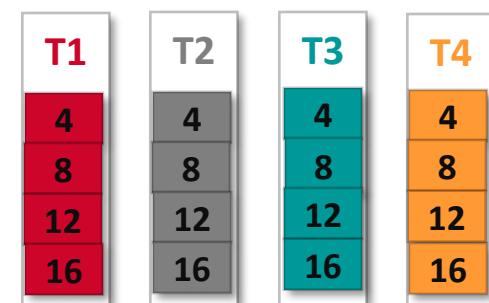
Input-only Parameters	
Parameter	Description
sendbuf	Starting address of send buffer
recvbuf	Starting address of recv buffer
type	Data type of buffer elements
count	Number of elements in the buffers
operation	Reduction operation to be performed (e.g. sum)
comm	Communicator handle

Input/Output Parameters	
Parameter	Description
recvbuf	Starting address of receive buffer

Sendbuf (Before)



Recvbuf (After)



Overview of the MVAPICH Project

- High Performance open-source MPI Library
- Support for multiple interconnects
 - InfiniBand, Omni-Path, Ethernet/iWARP, RDMA over Converged Ethernet (RoCE), AWS EFA, OPX, Broadcom RoCE, Intel Ethernet, Rockport Networks, Slingshot 10/11
- Support for multiple platforms
 - x86, OpenPOWER, ARM, Xeon-Phi, GPGPUs (NVIDIA and AMD)
- Started in 2001, first open-source version demonstrated at SC '02
- Supports the latest MPI-4.1 standard
- <http://mvapich.cse.ohio-state.edu>
- Additional optimized versions for different systems/environments:
 - MVAPICH-Plus (Unification of MVAPICH2-X and MVAPICH2-GDR), since 2023
 - MVAPICH2-X (Advanced MPI + PGAS), since 2011
 - MVAPICH2-GDR with support for NVIDIA (since 2014) and AMD (since 2020) GPUs
 - MVAPICH2-MIC with support for Intel Xeon-Phi, since 2014
 - MVAPICH2-Virt with virtualization support, since 2015
 - MVAPICH2-EA with support for Energy-Awareness, since 2015
 - MVAPICH2-Azure for Azure HPC IB instances, since 2019
 - MVAPICH2-X-AWS for AWS HPC+EFA instances, since 2019
- Tools:
 - OSU MPI Micro-Benchmarks (OMB), since 2003
 - OSU InfiniBand Network Analysis and Monitoring (INAM), since 2015



- Used by more than 3,450 organizations in 92 countries (listed under the Users Tab of the MVAPICH page)
- More than 1.93 Million downloads from the OSU site directly
- Empowering many TOP500 clusters (Nov '24 ranking)
 - 15th, 10,649,600-core (Sunway TaihuLight) at NSC, Wuxi, China
 - 52nd, 448, 448 cores (Frontera) at TACC
 - 72nd, 288,288 cores (Lassen) at LLNL
 - 91st, 570,020 cores (Nurion) in South Korea and many others
- Available with software stacks of many vendors and Linux Distros (RedHat, SuSE, OpenHPC, and Spack)
- Partner in the 52nd ranked TACC Frontera system
- Empowering Top500 systems for more than 20+ years

GPU-Aware (CUDA-Aware) MPI Library: MVAPICH2-GDR

- Standard MPI interfaces used for unified data movement
- Takes advantage of Unified Virtual Addressing (>= CUDA 4.0)
- Overlaps data movement from GPU with RDMA transfers

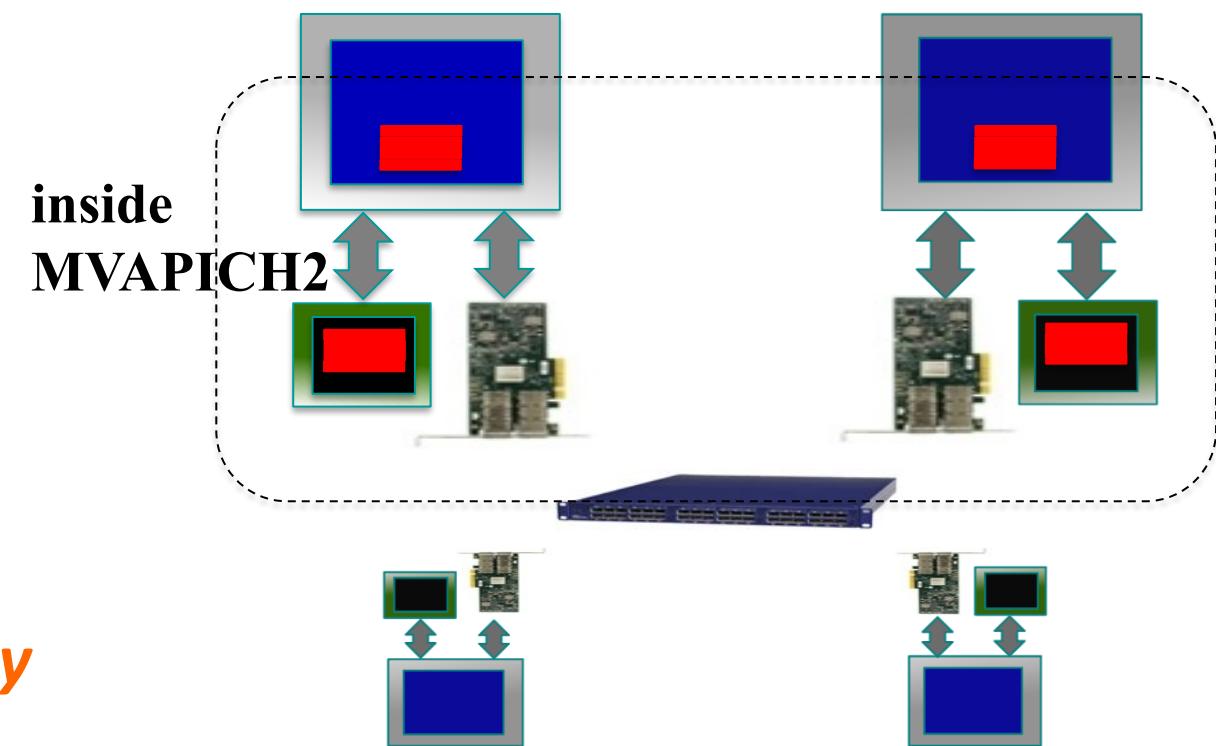
At Sender:

```
MPI_Send(s_devbuf, size, ...);
```

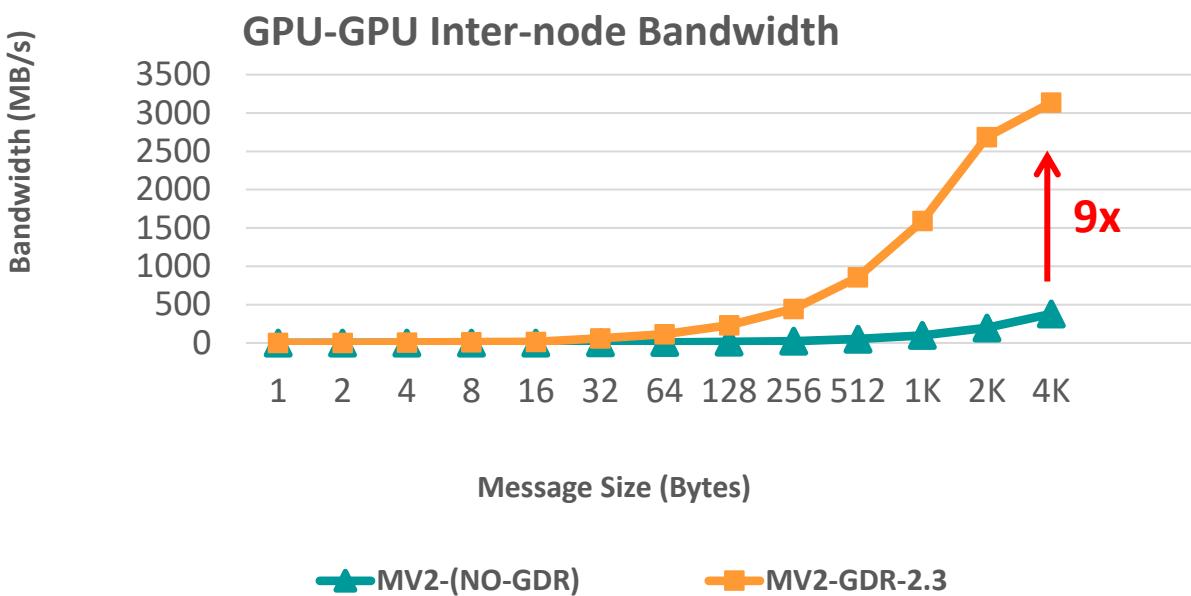
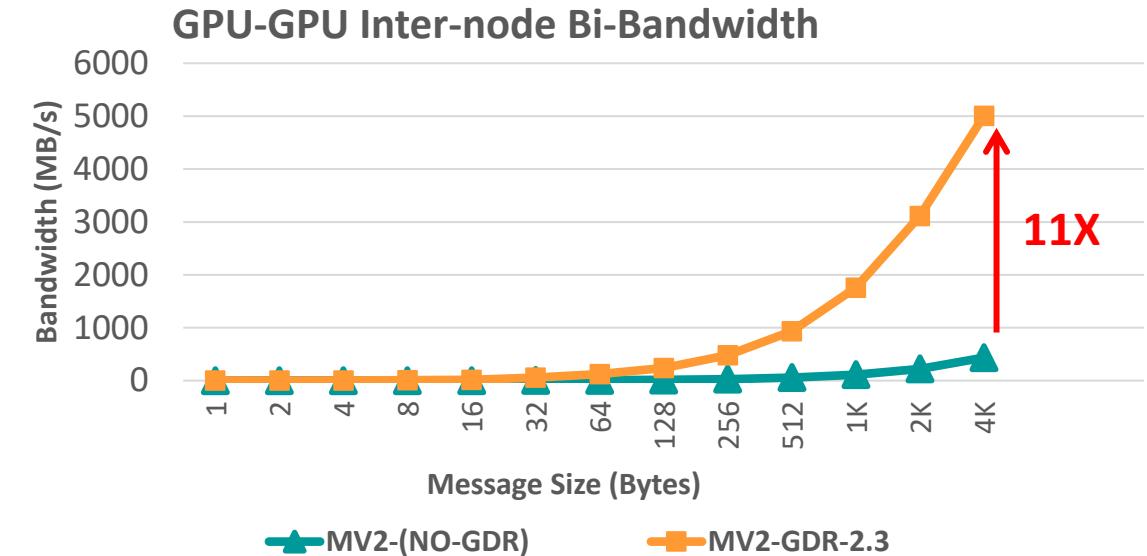
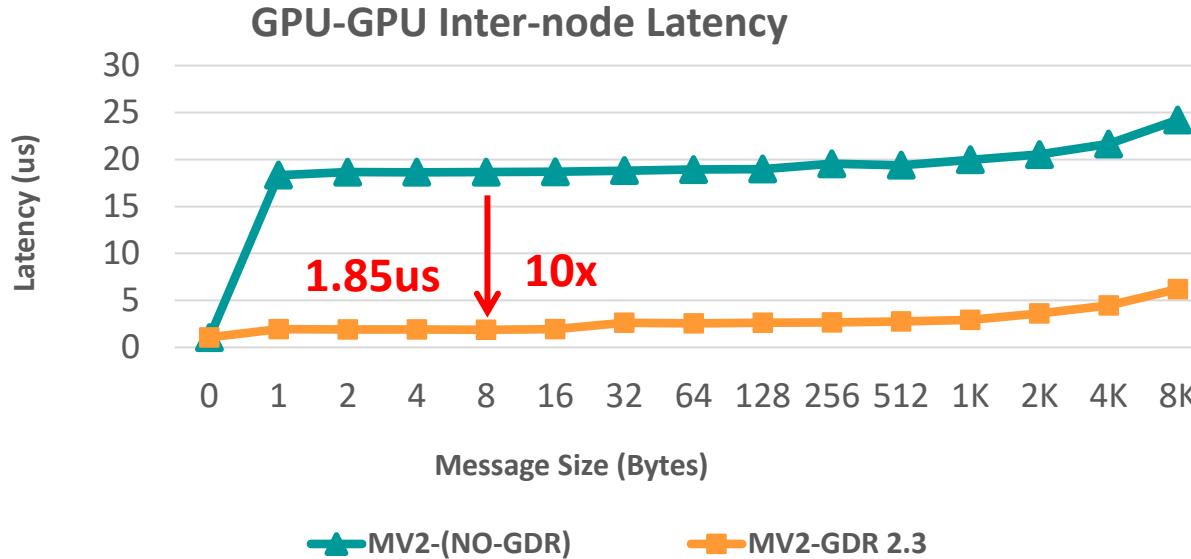
At Receiver:

```
MPI_Recv(r_devbuf, size, ...);
```

High Performance and High Productivity



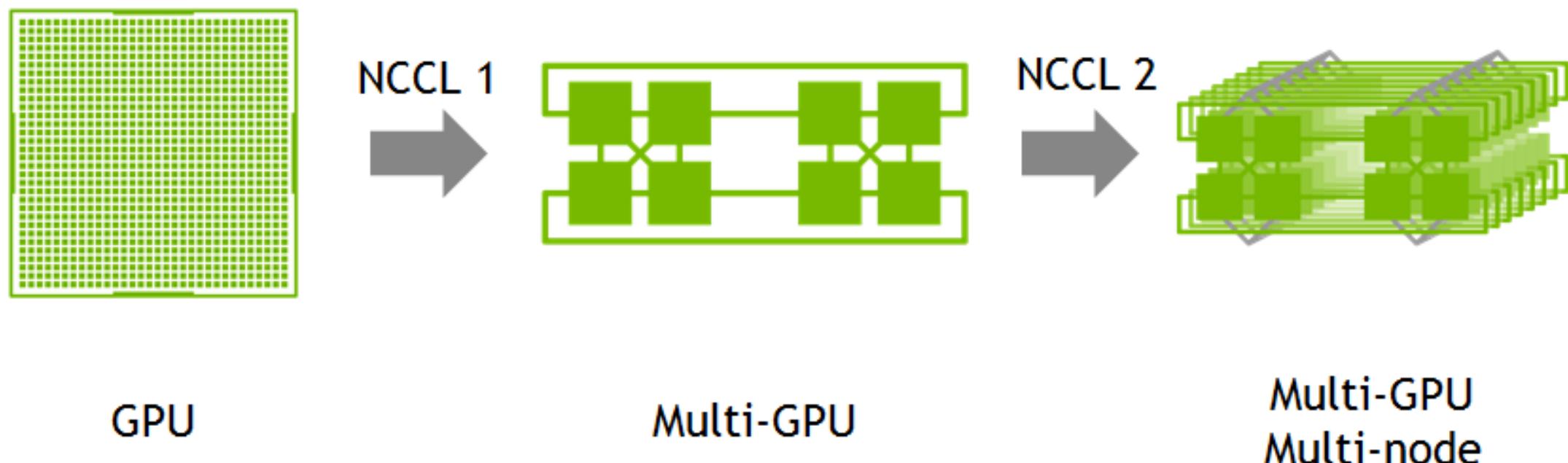
Optimized MVAPICH2-GDR Design



MVAPICH2-GDR-2.3.1
Intel Haswell (E5-2687W @ 3.10 GHz) node - 20 cores
NVIDIA Volta V100 GPU
Mellanox Connect-X4 EDR HCA
CUDA 9.0
Mellanox OFED 4.0 with GPU-Direct-RDMA

NCCL Communication Library

- NVIDIA Collective Communication Library (NCCL)
- Main Motivation: Deep Learning workloads
- NCCL1– efficient dense-GPU communication within the node
- NCCL2– multiple DGX systems connected to each other with InfiniBand systems



Courtesy: <https://developer.nvidia.com/nccl>

Outline

- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- **Challenges in Exploiting HPC Technologies for DL**
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

Broad Challenge: Exploiting HPC for Machine Learning/Deep Learning/Data Science Frameworks

*How to efficiently scale-out
Machine Learning (ML)/Deep Learning (DL)/Data
Science frameworks and take advantage of
heterogeneous
High Performance Computing (HPC) resources?*

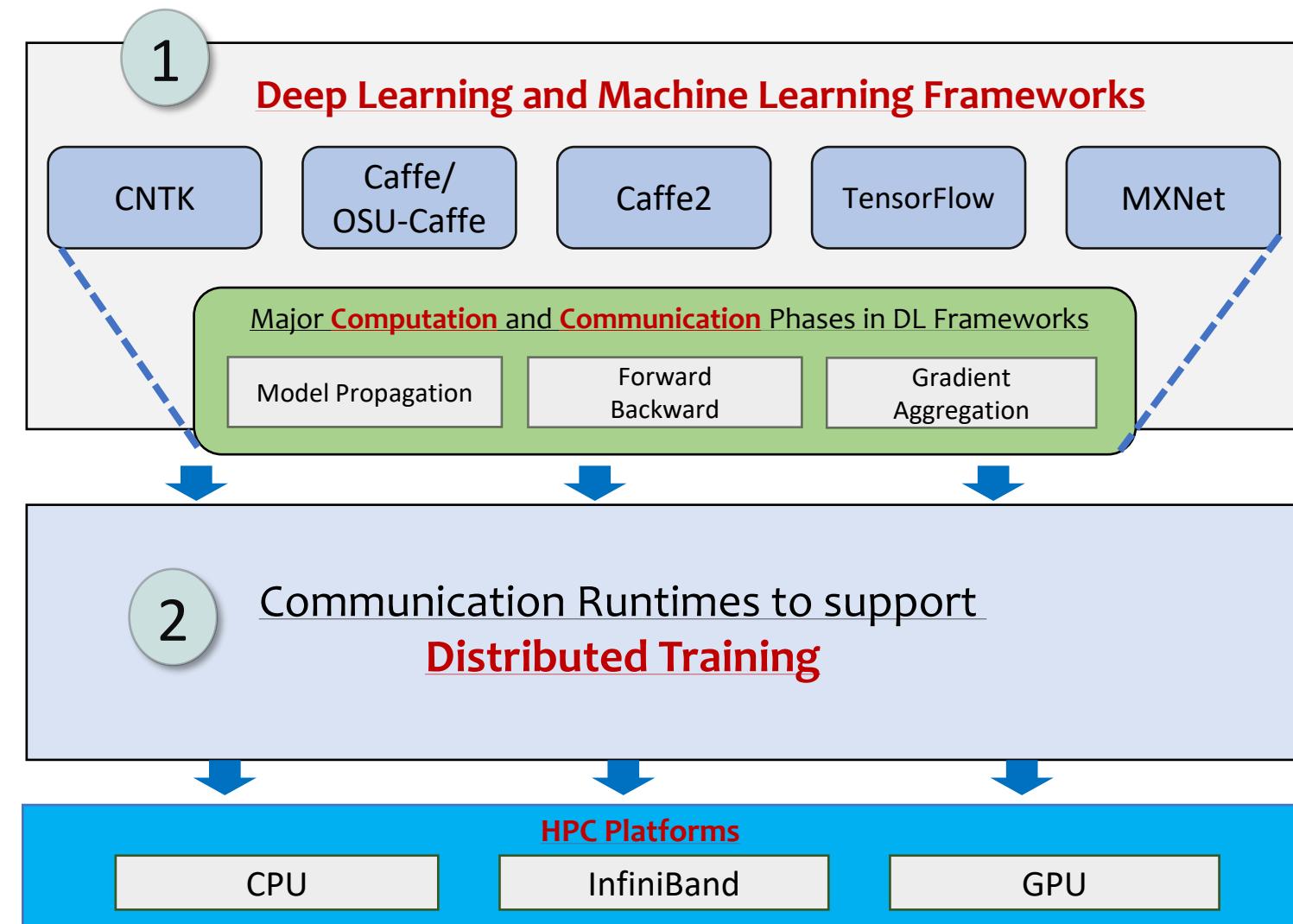
Research Challenges to Exploit HPC Technologies

1. What are the fundamental issues in designing **DL frameworks**?

- Memory Requirements
- **Computation** Requirements
- **Communication** Overhead

2. Why do we need to support **distributed training**?

- To overcome the limits of single-node training
- To better utilize hundreds of existing HPC Clusters



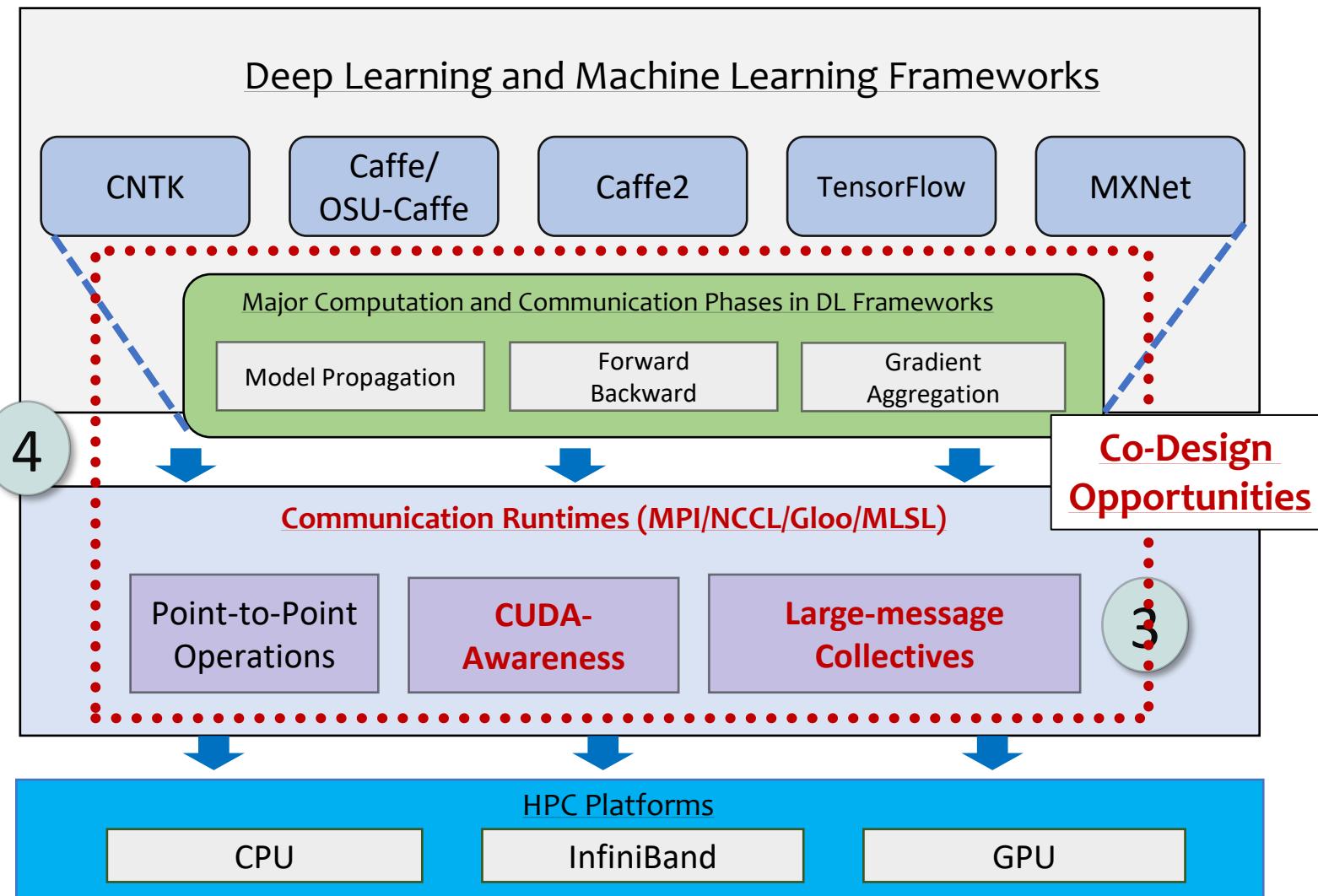
Research Challenges to Exploit HPC Technologies (Cont'd)

3. What are the **new design challenges** brought forward by DL frameworks for Communication runtimes?

- Large Message **Collective Communication** and Reductions
- GPU Buffers (**CUDA-Awareness**)

4. Can a **Co-design** approach help in achieving Scale-up and Scale-out efficiently?

- **Co-Design** the support at **Runtime level** and Exploit it at the **DL Framework level**
- What performance benefits can be observed?
- What needs to be fixed at the **communication runtime** layer?

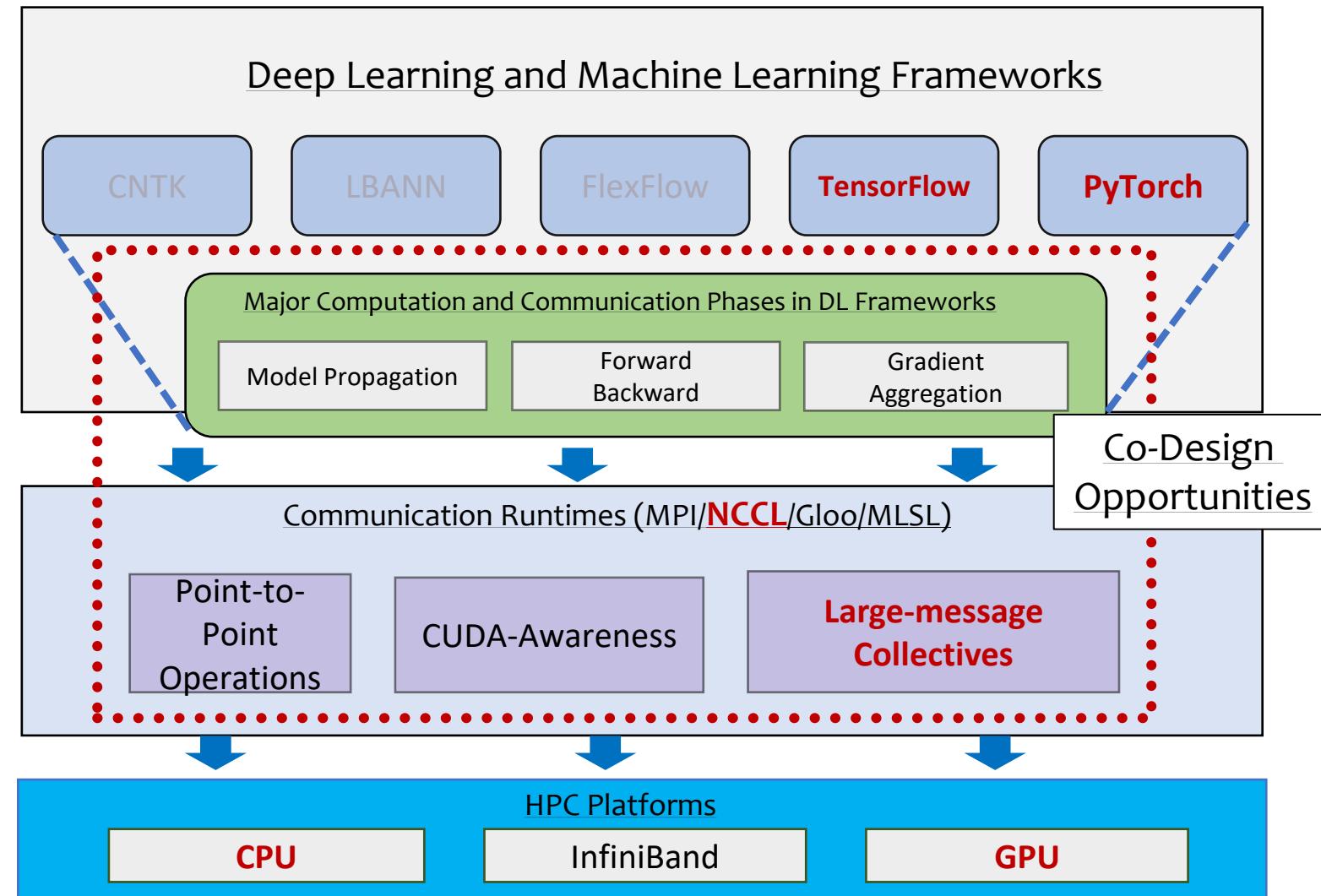


Outline

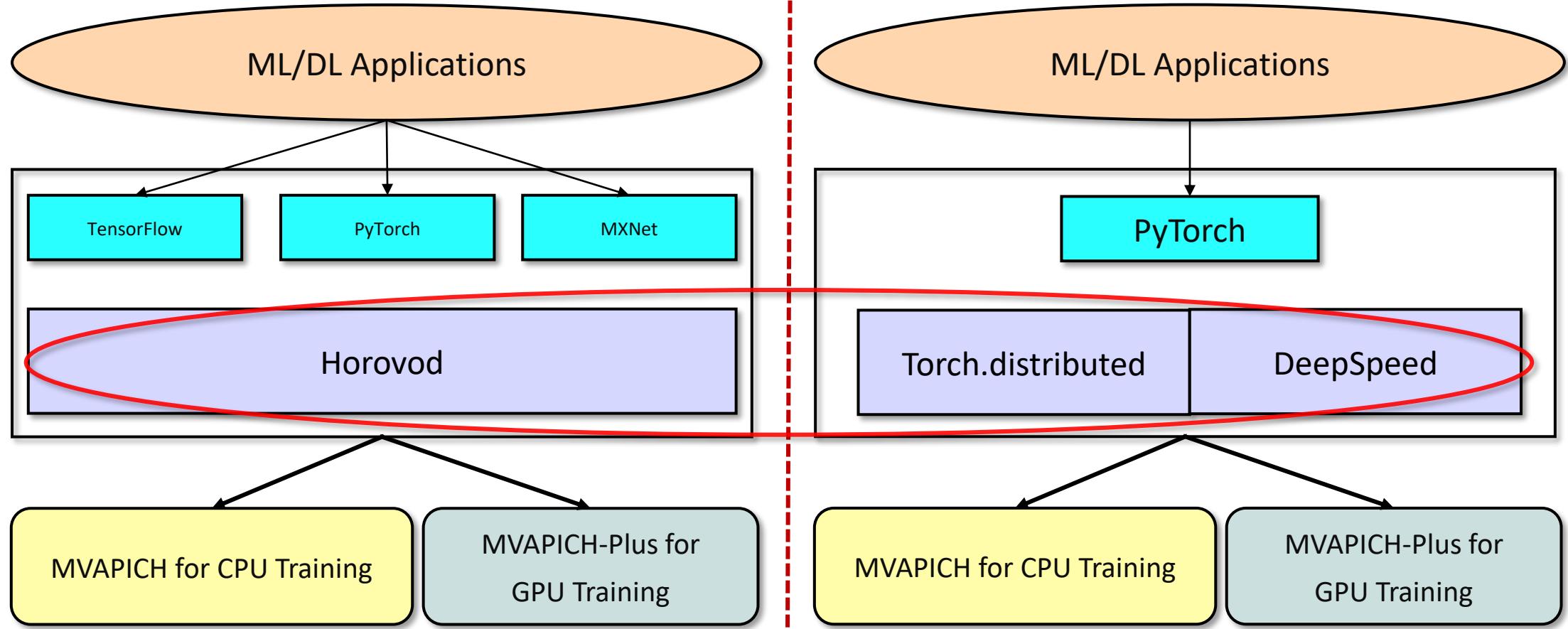
- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- **Advanced Distributed Training**
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- Conclusion

Solutions and Case Studies: Exploiting HPC for DL

- Data Parallelism
 - Distributed Training for TensorFlow and PyTorch
 - AccDP
- Model and Hybrid Parallelism
 - ZeRO
 - 3D Parallelism



MVAPICH (MPI)-driven Infrastructure for ML/DL Training: MPI4DL



More details available from: <https://github.com/OSU-Nowlab/pytorch/tree/hidl-2.0> and <http://hidl.cse.ohio-state.edu>

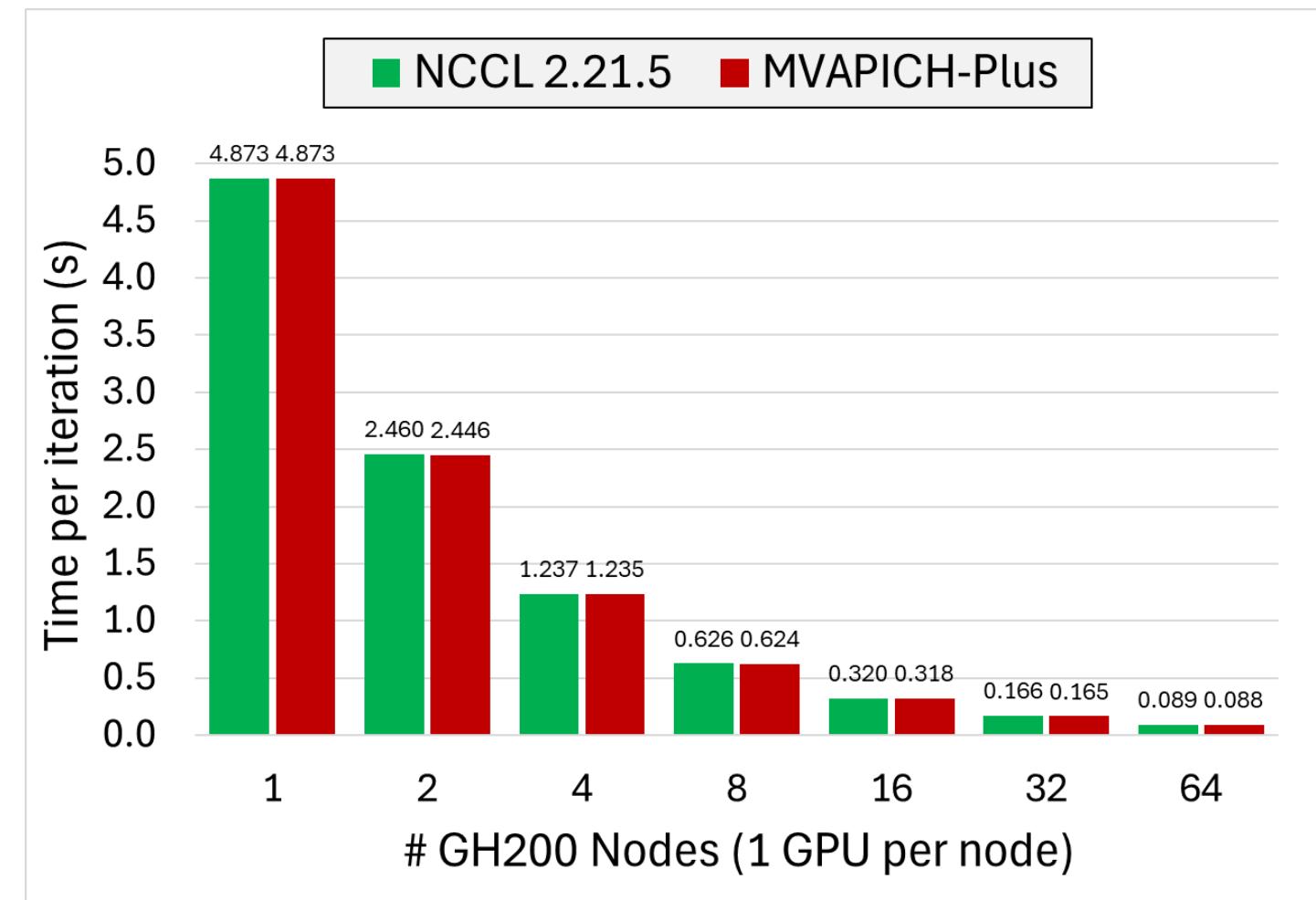
HiDL 2.0 Release

- Support for PyTorch 2.7.1 and later versions
- Full support for PyTorch Native DDP training
- Support for optimized MPI communication
 - Efficient large-message collectives (e.g., Allreduce) on various CPUs and GPUs
 - GPU-Direct Ring and Two-level multi-leader algorithms for Allreduce operations
 - Support for fork safety in distributed training environments
 - Exploits efficient large message collectives in MVAPICH-Plus 4.0 and later
- Open-source PyTorch version with advanced MPI backend support - Available in our PyTorch tag
- Vendor-neutral stack with competitive performance and throughput to GPU-based collective libraries
- Tested on modern HPC clusters (etc, OLCF Frontier, TACC Vista) with up-to-date accelerator generations (etc. AMD NVIDIA)
- Compatible with
 - InfiniBand Networks: Mellanox InfiniBand adapters (EDR, FDR, HDR, NDR)
 - Slingshot Networks: HPE Slingshot
 - GPU&CPU Support:
 - NVIDIA GPU A100, H100, GH200
 - AMD MI200 series GPUs
 - Software Stack:
 - CUDA [12.x] and Latest CuDNN
 - ROCm [6.x]
 - (NEW)PyTorch [2.7.1]
 - (NEW)Python [3.x]

More details available from: <https://github.com/OSU-Nowlab/pytorch/tree/hidl-2.0>
and <http://hidl.cse.ohio-state.edu>

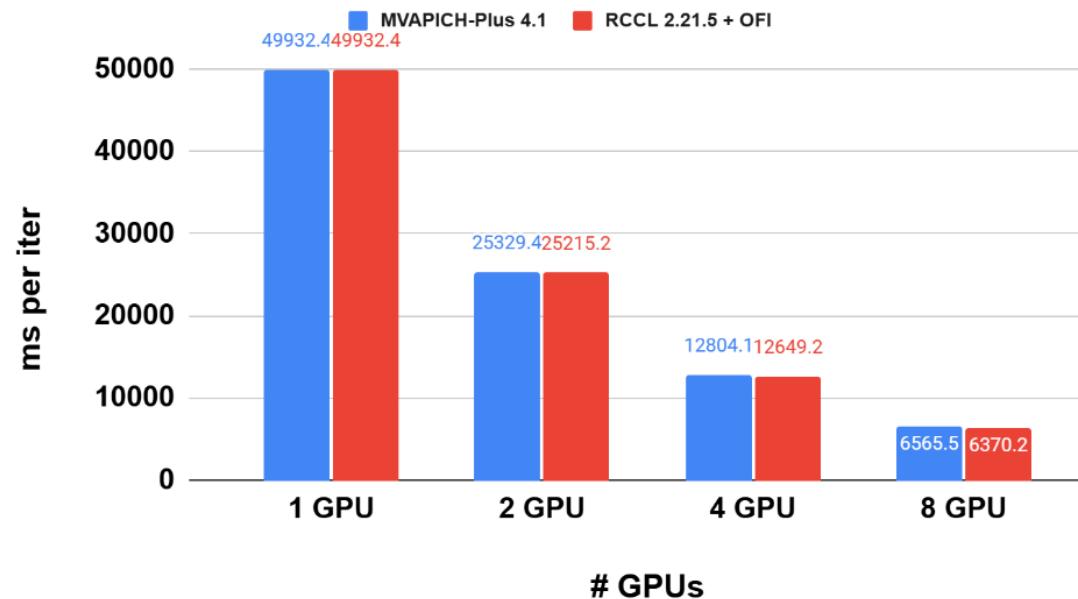
Distributed Data Parallel Training on GH200 (Vista)

- Torch Distributed
- Application: GPT-2 model training using nanoGPT.
- Hardware: Vista System @TACC
 - GH200 Superchips each with:
 - 72 ARM cores with 120 GB LPDDR.
 - H100 GPU with 96GB HBM3.
 - NVIDIA NDR InfiniBand (400Gb/s)
- Software:
 - PyTorch 2.6.0
 - NCCL 2.21.5
 - MVAPICH-Plus 4.1

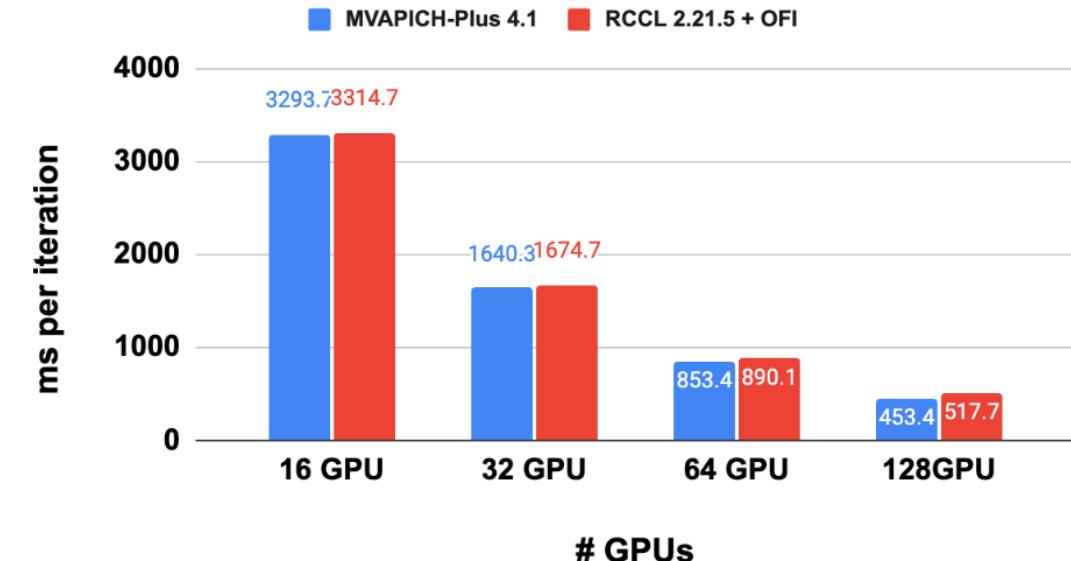


Distributed Data Parallel Training (Frontier)

GPT-2 DDP 1.5 Million Token per Iter



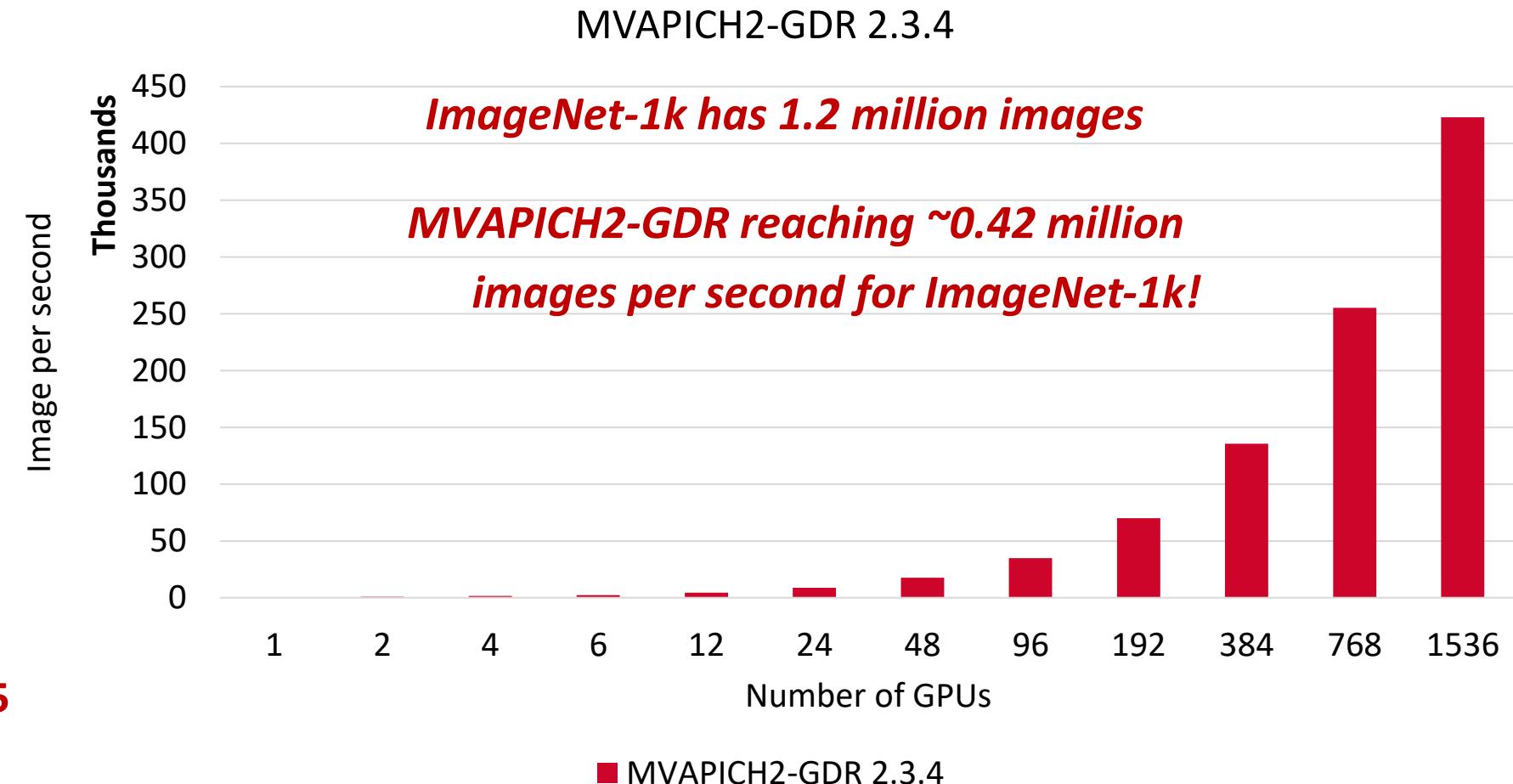
GPT-2 DDP 1.5 Million Token per Iter



- End-to-end GPT-2 Training with Openwebtext using Distributed Data Parallel
- **12.4%** less ms per iteration (compared to RCCL 2.21.5 + OFI) for 128 GPUs

Distributed TensorFlow on ORNL Summit (1,536 GPUs)

- ResNet-50 Training using TensorFlow benchmark on SUMMIT -- 1536 Volta GPUs!
- 1,281,167 (1.2 mil.) images
- Time/epoch = 3 seconds
- Total Time (90 epochs)
 $= 3 \times 90 = 270$ seconds = **4.5 minutes!**

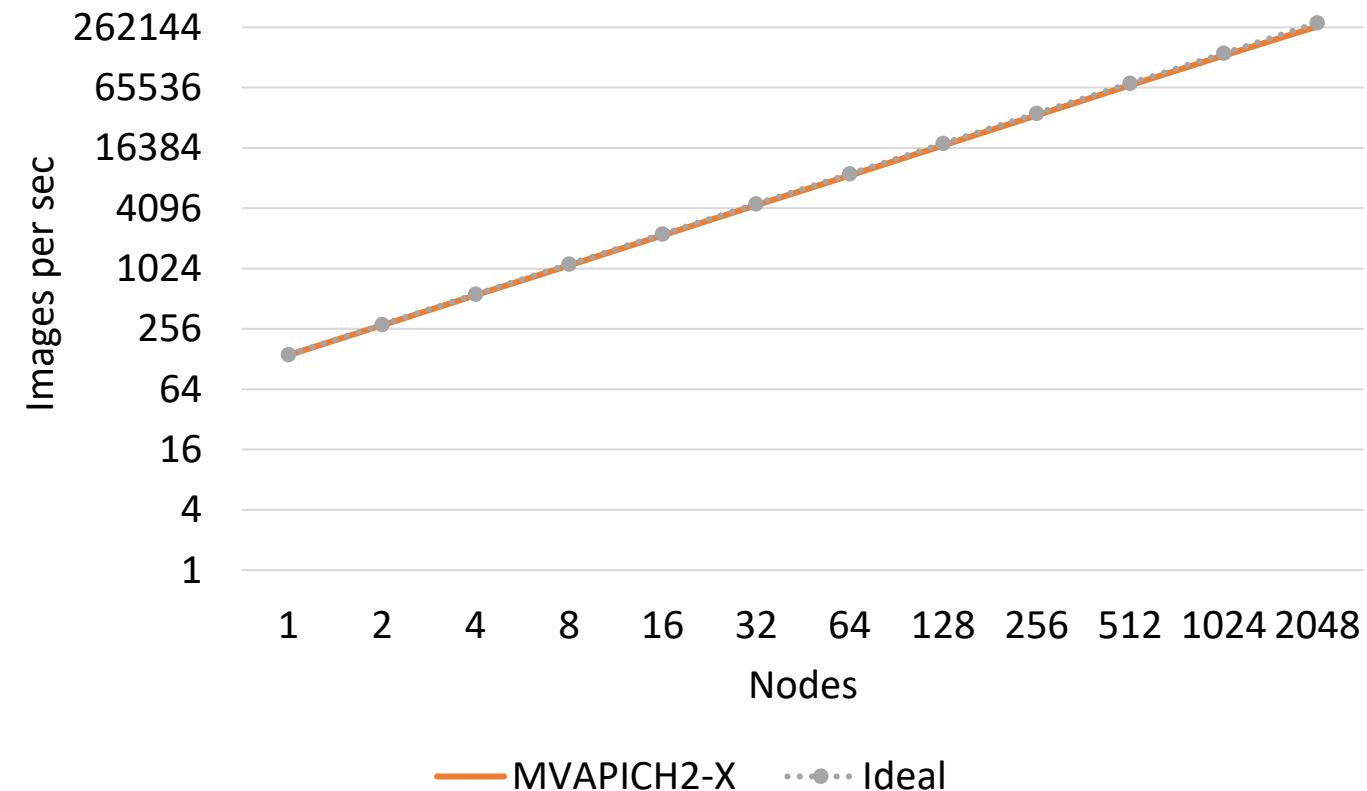


*We observed issues for NCCL2 beyond 384 GPUs

Platform: The Summit Supercomputer (#2 on Top500.org) – 6 NVIDIA Volta GPUs per node connected with NVLink, CUDA 10.1

Distributed TensorFlow on TACC Frontera (2048 CPU nodes)

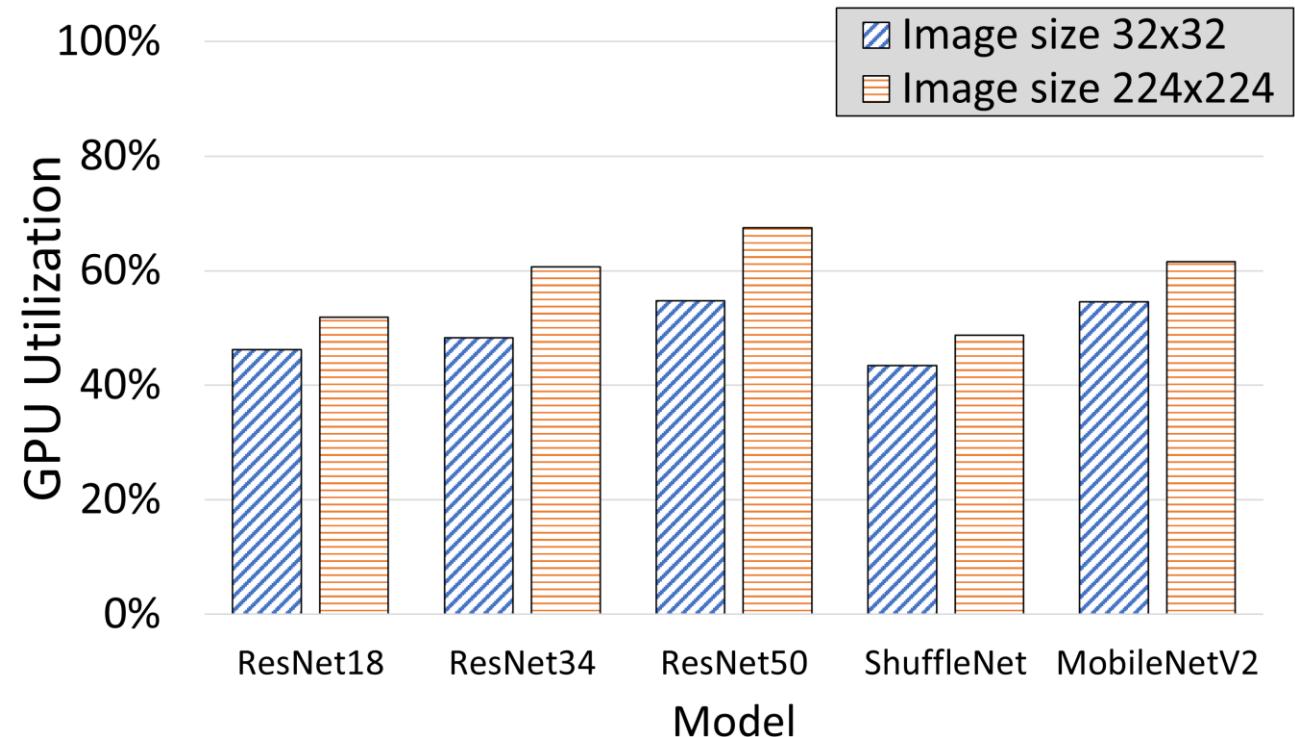
- Scaled TensorFlow to 2048 nodes on Frontera using MVAPICH2 and IntelMPI
- MVAPICH2 delivers close to the ideal performance for DNN training
- Report a peak of **260,000 images/sec** on 2048 nodes
- On 2048 nodes, ResNet-50 can be trained in **7 minutes!**



A. Jain, A. A. Awan, H. Subramoni, DK Panda, "Scaling TensorFlow, PyTorch, and MXNet using MVAPICH2 for High-Performance Deep Learning on Frontera", DLS '19 (SC '19 Workshop).

AccDP: GPU Utilization for DNN Training

- Modern GPUs are computational workhorses in HPC systems and are used in parallel to reduce DNN training time.
- However, GPUs are not fully utilized by DNN training workloads especially for small-to-medium DL models and/or input size.
- The figure shows the resource utilization of NVIDIA A100 GPU during the training phase of different DNN models with two input sizes. (We choose the largest possible batch sizes for best performance)
- We observed a utilization as low as **43%** for ResNet18 with 32x32 input size to **63%** for ResNet50 with image size 224x224.



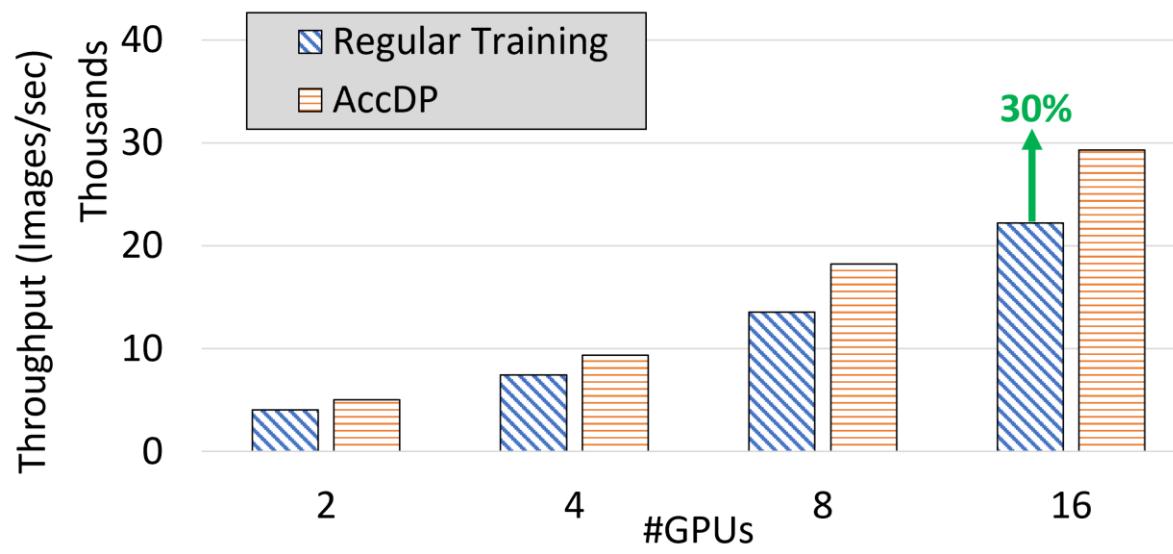
NVIDIA A100 GPU utilization during DNN training of different models with different input sizes

N. Alnaasan, A. Jain, A. Shafi, H. Subramoni, and DK Panda, “AccDP: Accelerated Data-Parallel Distributed DNN Training for Modern GPU-Based HPC Clusters”, HiPC’22.

AccDP: Performance Improvement

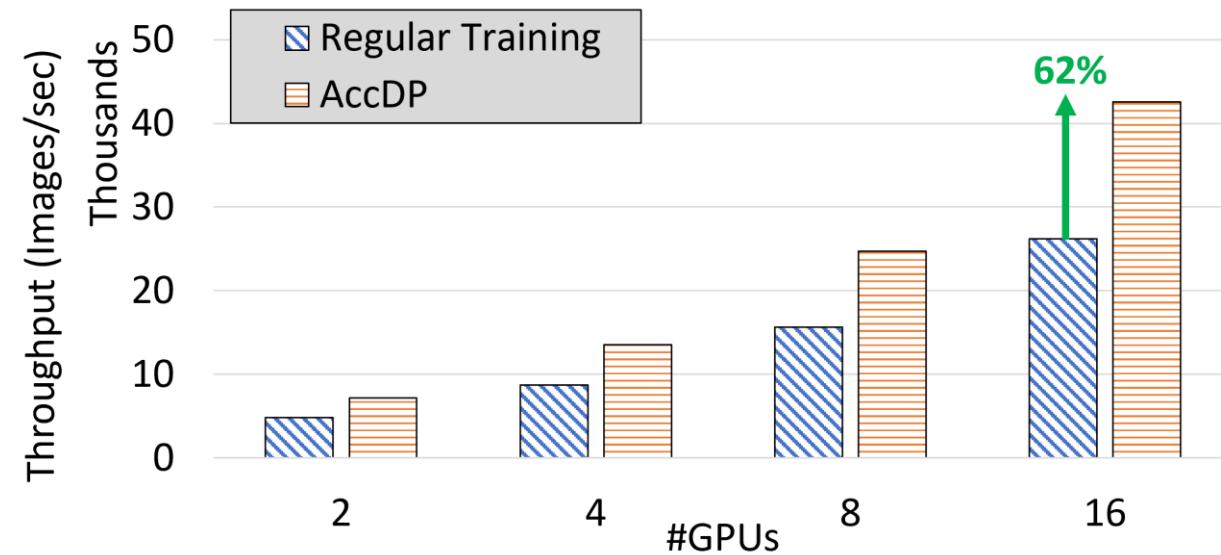
Multi node with ResNet18

- ResNet18 training throughput comparison between regular training and AccDP (proposed design) for different DNN models on up to 8 nodes 2 GPUs per node (16 GPUs) with 4 MPS clients per GPU



Multi node with ShuffleNet

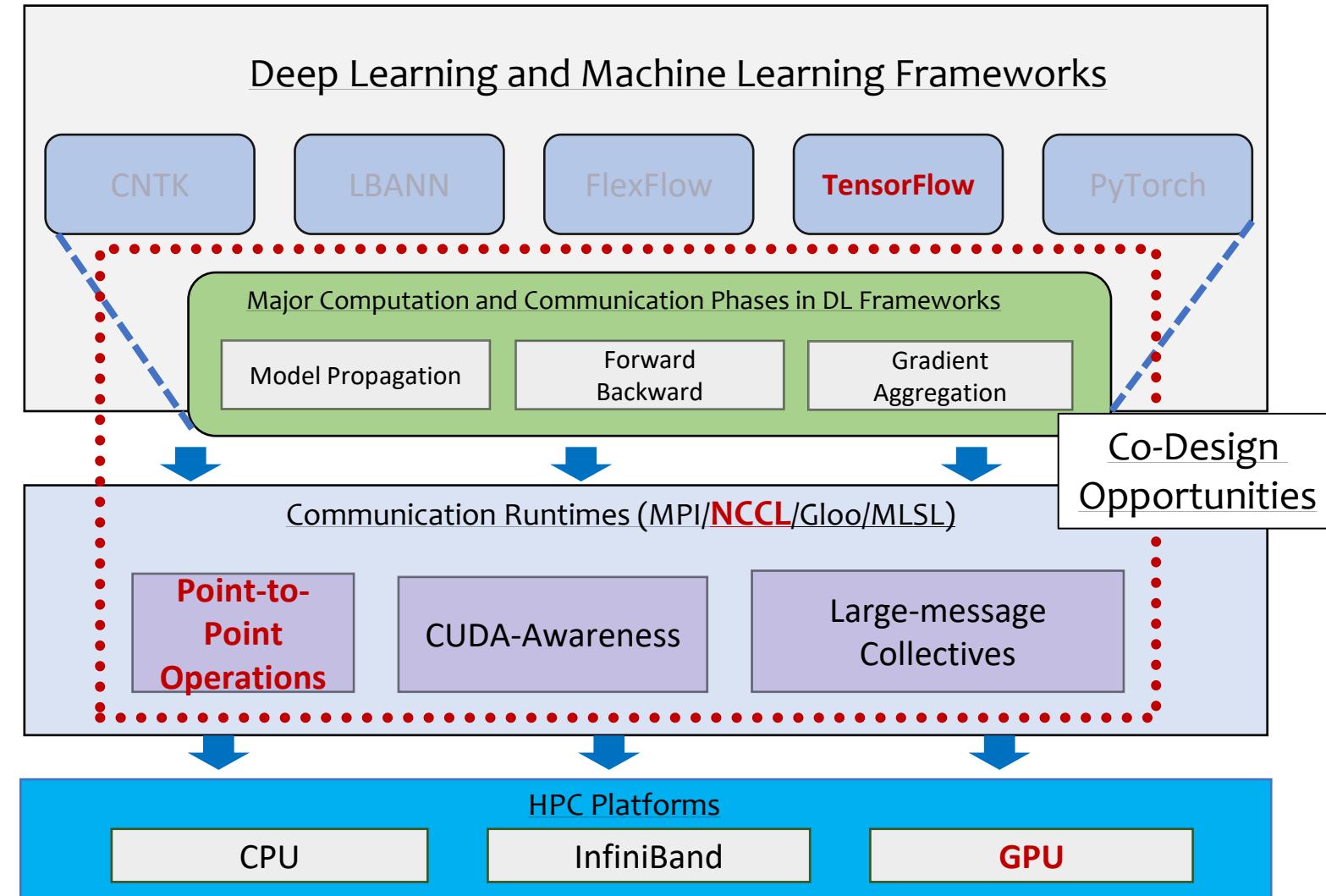
- ShuffleNet training throughput comparison between regular training and AccDP (proposed design) for different DNN models on up to 8 nodes 2 GPUs per node (16 GPUs) with 4 MPS clients per GPU.



N. Alnaasan, A. Jain, A. Shafi, H. Subramoni, and DK Panda, "AccDP: Accelerated Data-Parallel Distributed DNN Training for Modern GPU-Based HPC Clusters", HiPC'22.

Solutions and Case Studies: Exploiting HPC for DL

- Data Parallelism
 - Distributed Training for TensorFlow and PyTorch
 - AccDP
- Model and Hybrid Parallelism
 - ZeRO
 - 3D Parallelism



ZeRO 4-way data parallel training

Using:

- P_{os} (Optimizer state)
- P_g (Gradient)
- P_p (Parameters)

Courtesy: <https://www.microsoft.com/en-us/research/blog/zero-deepspeed-new-system-optimizations-enable-training-models-with-over-100-billion-parameters/>

Memory Anatomy of a DNN (for ZeRO/FSDP)

Key question: What is the GPU memory M required to fit a model during training:

$$M_{Tot} = M_m + M_o + M_g + M_a$$

Where M_m is model memory, M_o optimizer memory, M_g gradient memory, and M_a activation memory

- p : Num model parameters
- k_l : Low precision B/param
- k_h : High precision B/param
- d : GPU Devices
- s : Sequence length
- b : Batch size
- h : Hidden size
- L : Transformer layers
- a : Num attention heads
- z : ZeRO stage

$$M_m = \begin{cases} \frac{k_l \cdot p}{d}, & z = 3 \\ k_l \cdot p, & \text{else} \end{cases}$$

$$M_o = \begin{cases} \frac{(3k_h) \cdot p}{d}, & z \geq 1 \\ (3k_h) \cdot p, & \text{else} \end{cases}$$

$$M_g \leq \begin{cases} \frac{(k_l \text{ or } k_h) \cdot p}{d}, & z \geq 2 \\ (k_l \text{ or } k_h) \cdot p, & \text{else} \end{cases}$$

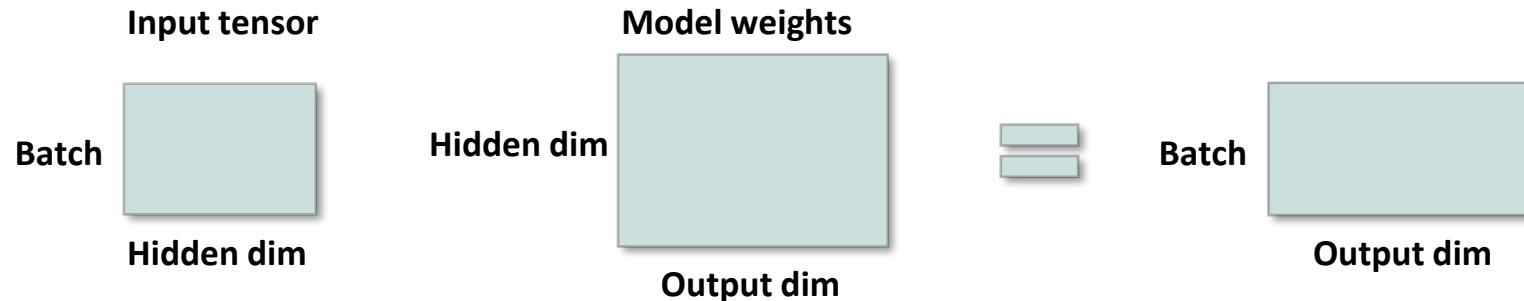
$$M_a \approx sbhL([16k_l + 2] + [2k_l + 1]\frac{a \cdot s}{h})$$

DeepSpeed ZeRO

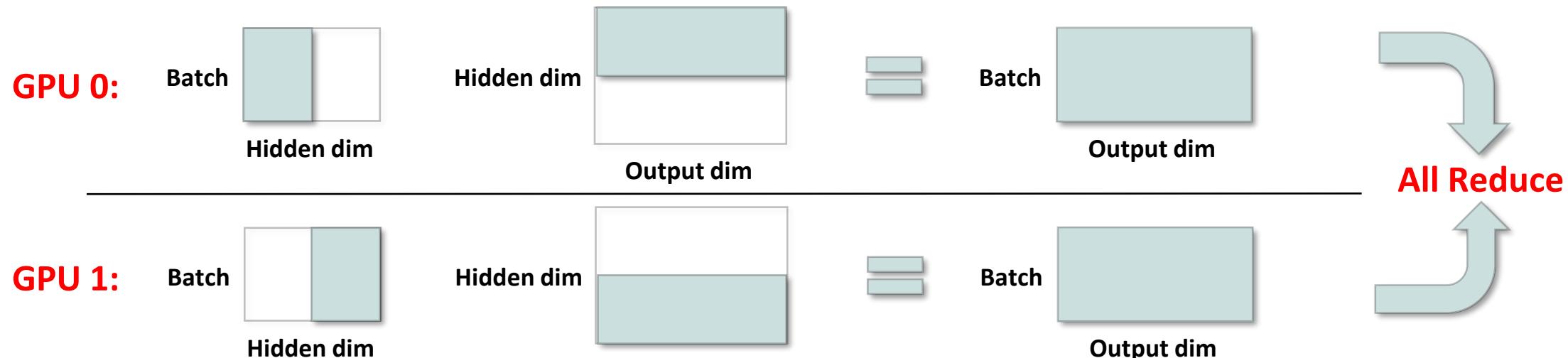
- Instead of being limited by the **device** memory, we are now limited by the **aggregate** memory
- E.g. You want to train a trillion-parameter model on 1024 GPUs with 16 GB memory each
 - With 16-bit precision, model+optimizer = ~16 TB of memory
 - We can fit this into 1024 GPUs with ZeRO: $\frac{16 \text{ TB}}{1024 \text{ GPUs}} = 16 \frac{\text{GB}}{\text{GPU}}$
- ZeRO-Infinity introduces offload to CPU memory or NVMe disk for the truly desperate
- Since ZeRO removes the DP memory limit, do we still need MP?
 - There are still models and data samples (e.g. pathology, astronomy, etc) that don't fit inside GPU memory **even with ZeRO**
 - We can use pipeline + tensor parallelism along with ZeRO for these cases (called 3D-parallel, more on this later!)

Tensor Parallelism

- LLM models consist of matrix multiplications.

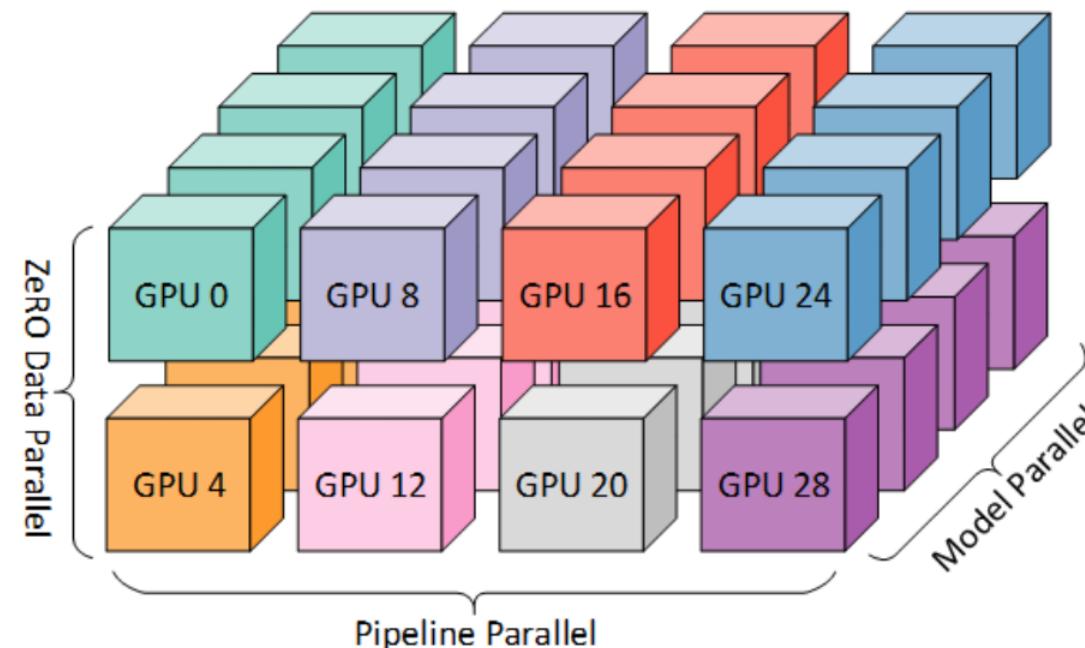


- Tensor Parallelism splits along hidden dim, and distributes the computation to multiple GPUs.



3D Parallelism

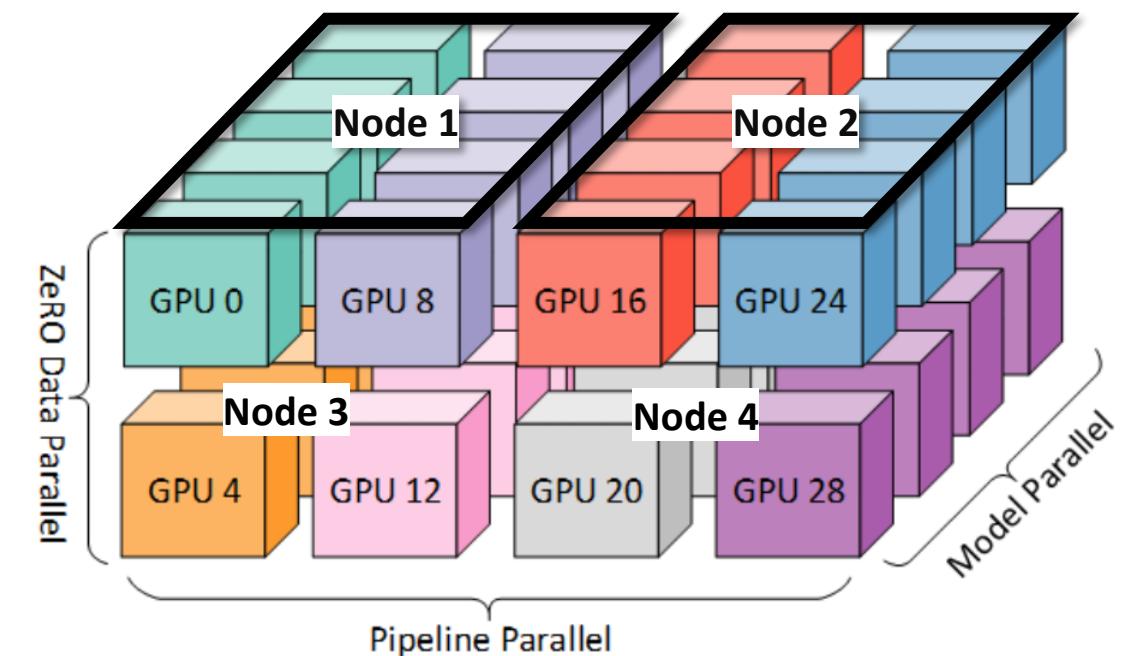
- Combine PP with TP and DP for 3D parallelism. For example:
 - Split given layer(s) via TP across 4 GPUs
 - Split the model into 4 pipeline stages
 - The above TP+PP combination compose a single DP unit
 - Use 2 DP units with the above configuration for 32-GPU parallelism
- **Question:** Given that each node contains 8 GPUs, where should you place the node boundaries?



Credit: <https://www.microsoft.com/en-us/research/blog/deepspeed-extreme-scale-model-training-for-everyone/>

3D Parallelism

- Combine PP with TP and DP for 3D parallelism. For example:
 - Split given layer(s) via TP across 4 GPUs
 - Split the model into 4 pipeline stages
 - The above TP+PP combination compose a single DP unit
 - Use 2 DP units with the above configuration for 32-GPU parallelism
- **Question:** Given 4 nodes with 8 GPUs each, where should you place the node boundaries?
- **Answer:** Keep as many TP partitions as possible within a node
 - Each model replica requires $TP \times PP = 16$ GPUs
 - Two pipeline stages per node
 - Pipeline-parallel pt2pt comms across nodes, no inter-node TP
 - Between pipeline stages 2 and 3 out of the total 4
 - ZeRO-1 across nodes as well, but same comms volume as DP and easy to overlap with compute



Lab 2 – Out-of-core DNN Training using DeepSpeed

- Objectives
 - Test an out-of-core DNN on a single node (BERT 2.5B)
 - Train the out-of-core DNN on two node using DeepSpeed
- Tasks
 - Task 1: Single GPU
 - Task 2: Multi-GPU

Lab 2 – Task 1: Test a 2.5B Bert DNN on a single GPU

```
$ cd /opt/tutorials/hot1-hidl-tutorial/lab2
$ srun -N 1 -p bdw-v100 train-bert-single.sh

+ /opt/tutorials/hidl-env/miniconda3/envs/deepspeed/bin/deepspeed -H /tmp/hosts_425272 /opt/tutorials/hidl-
env/deepspeed_benchmarks/train_bert.py --checkpoint_dir /tmp/checks --num_layers 192 --ff_dim 4096 --h_dim 1024 --batch_size
1 --num_iterations 10

Traceback (most recent call last):
  File "/opt/tutorials/hidl-env/deepspeed_benchmarks/train_bert.py", line 791, in <module>
    fire.Fire(train)
  File "/opt/tutorials/hidl-env/miniconda3/envs/deepspeed/lib/python3.10/site-packages/fire/core.py", line 141, in Fire
    component_trace = _Fire(component, args, parsed_flag_args, context, name)
  File "/opt/tutorials/hidl-env/miniconda3/envs/deepspeed/lib/python3.10/site-packages/fire/core.py", line 466, in _Fire
    component, remaining_args = _CallAndUpdateTrace(
  File "/opt/tutorials/hidl-env/miniconda3/envs/deepspeed/lib/python3.10/site-packages/fire/core.py", line 681, in _CallAndUpdateTrace
    component = fn(*varargs, **kwargs)
  File "/opt/tutorials/hidl-env/deepspeed_benchmarks/train_bert.py", line 759, in train
    optimizer.step()
  File "/opt/tutorials/hidl-env/labs/lab4/pytorch/torch/optim/optimizer.py", line 391, in wrapper
    out = func(*args, **kwargs)
  File "/opt/tutorials/hidl-env/labs/lab4/pytorch/torch/optim/optimizer.py", line 76, in _use_grad
    ret = func(self, *args, **kwargs)
  File "/opt/tutorials/hidl-env/labs/lab4/pytorch/torch/optim/adam.py", line 159, in step
    has_complex = self._init_group(
  File "/opt/tutorials/hidl-env/labs/lab4/pytorch/torch/optim/adam.py", line 115, in _init_group
    state['exp_avg_sq'] = torch.zeros_like(p, memory_format=torch.preserve_format)
torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 16.00 MiB. GPU
```

Lab 2 – Task 2: Run 2.5B Bert DNN on two GPUs

```
$ cd /opt/tutorials/hot1-hidl-tutorial/lab2
$ srun -N 2 --reservation=dltutorial run_bert.sh

+ /opt/tutorials/hidl-env/miniconda3/envs/deepspeed/bin/deepspeed -H /tmp/hosts_425274 /opt/tutorials/hidl-
env/deepspeed_benchmarks/train_bert_ds.py --checkpoint_dir /opt/tutorials/hidl-env/checks --num_layers 192 --ff_dim 4096 --
h_dim 1024 --batch_size 1 --num_iterations 10

gpu01: [2022-12-31 22:50:20,415] [INFO] [timer.py:197:stop] 0/4, RunningAvgSamplesPerSec=1.989045629615473, CurrSamplesPerSec=1.9197348998894654,
MemAllocated=18.42GB, MaxMemAllocated=25.43GB
gpu01: [2022-12-31 22:50:21,470] [INFO] [stage_1_and_2.py:1765:step] [deepspeed] OVERFLOW! Rank 0 Skipping step. Attempted loss scale: 268435456.0,
reducing to 134217728.0
gpu01: [2022-12-31 22:50:21,472] [INFO] [timer.py:197:stop] 0/5, RunningAvgSamplesPerSec=1.9581760061302556, CurrSamplesPerSec=1.8992247658347254,
MemAllocated=18.42GB, MaxMemAllocated=25.44GB
gpu01: [2022-12-31 22:50:22,518] [INFO] [stage_1_and_2.py:1765:step] [deepspeed] OVERFLOW! Rank 0 Skipping step. Attempted loss scale: 134217728.0,
reducing to 67108864.0
gpu01: [2022-12-31 22:50:22,520] [INFO] [timer.py:197:stop] 0/6, RunningAvgSamplesPerSec=1.9472242517233995, CurrSamplesPerSec=1.9150918757408228,
MemAllocated=18.42GB, MaxMemAllocated=25.44GB
gpu01: [2022-12-31 22:50:23,557] [INFO] [stage_1_and_2.py:1765:step] [deepspeed] OVERFLOW! Rank 0 Skipping step. Attempted loss scale: 67108864.0,
reducing to 33554432.0
gpu01: [2022-12-31 22:50:23,558] [INFO] [timer.py:197:stop] 0/7, RunningAvgSamplesPerSec=1.9440531488041186, CurrSamplesPerSec=1.9314713530693848,
MemAllocated=18.42GB, MaxMemAllocated=25.44GB
gpu01: [2022-12-31 22:50:24,577] [INFO] [stage_1_and_2.py:1765:step] [deepspeed] OVERFLOW! Rank 0 Skipping step. Attempted loss scale: 33554432.0,
reducing to 16777216.0
gpu01: [2022-12-31 22:50:24,579] [INFO] [timer.py:197:stop] 0/8, RunningAvgSamplesPerSec=1.9473977612894298, CurrSamplesPerSec=1.9642949469669437,
MemAllocated=18.42GB, MaxMemAllocated=25.44GB
gpu01: [2022-12-31 22:50:25,644] [INFO] [stage_1_and_2.py:1765:step] [deepspeed] OVERFLOW! Rank 0 Skipping step. Attempted loss scale: 16777216.0,
reducing to 8388608.0
gpu01: [2022-12-31 22:50:25,645] [INFO] [timer.py:197:stop] 0/9, RunningAvgSamplesPerSec=1.9380716205436017, CurrSamplesPerSec=1.883938232505326,
MemAllocated=18.42GB, MaxMemAllocated=25.44GB
```

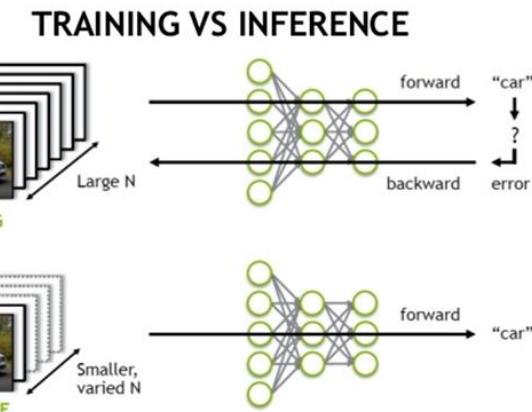
Outline

- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- **Distributed Inference Solutions**
- Open Issues and Challenges
- Conclusion

What is Deep Learning Inference?

- Deep learning Training & Inference

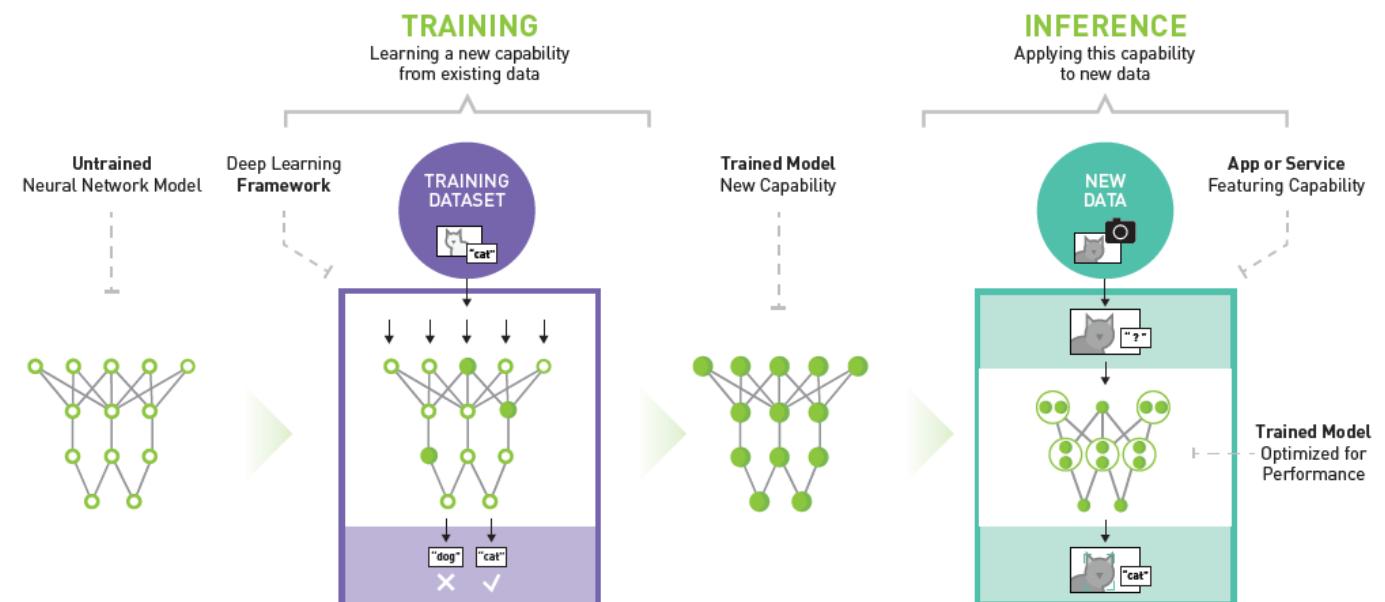
	Phase	Sensitivity
Training	Model-learning	Throughput
Inference	User-facing	Latency



- Inference: Latency-sensitive

- Final Phase of Deep Learning
- The closest end to users**

- Smaller batch size in the workflow
- User-end requests arrive randomly
- No need for model weights update
- Response time is the most crucial



Courtesy: <https://developer.nvidia.com/blog/nvidia-deep-learning-inference-platform-performance-study/>; <https://www.exxactcorp.com/blog/HPC/discover-the-difference-between-deep-learning-training-and-inference>

Inference Scenarios

1. Online vs. batch inference:

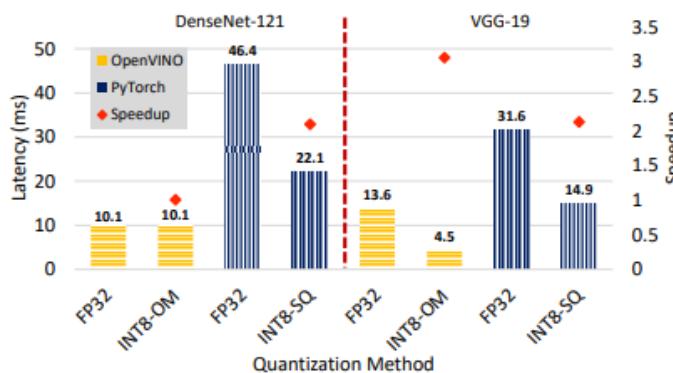
- Online Inference: used when real-time predictions are required
 - **Latency:** Lower latency is critical for real-time applications, and online inference focuses on minimizing the time it takes to process individual instances.
- Batch Inference: employed for processing large volumes of data at once
 - **Throughput:** Batch inference focuses on maximizing throughput by processing many instances simultaneously, rather than prioritizing latency.

2. Edge vs. HPC/Cloud inference:

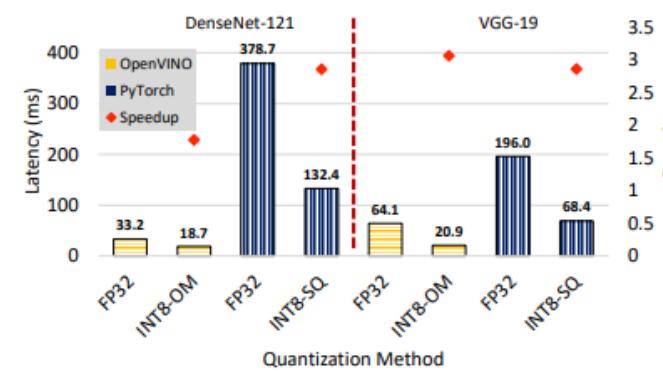
- Inference on the Edge: limited resources and require low-latency responses
 - **Latency:** Low-latency responses are crucial in edge scenarios, as real-time predictions may be necessary for applications like autonomous vehicles or IoT devices.
- Cloud Inference: more resources and better scalability
 - **Throughput:** HPC/cloud systems can scale horizontally and vertically, allowing for increased throughput when processing large volumes of data.

Quantization for DNN Inference on the Edge

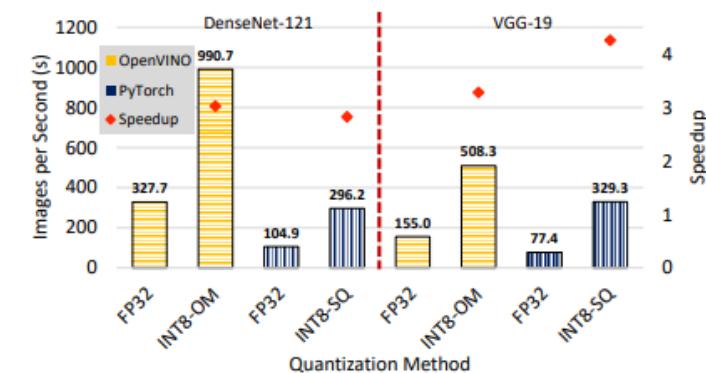
- Quantization uses FP16, INT16, and INT8 datatypes instead of FP32 to represent the weights and activations of DNN models.
- Using smaller datatypes to represent a model can lead to reduced memory footprint, smaller latency, and improved throughput.
- The quantization approach is especially beneficial for edge devices with limited memory and compute resources.



(a) Single-stream



(b) Multi-stream



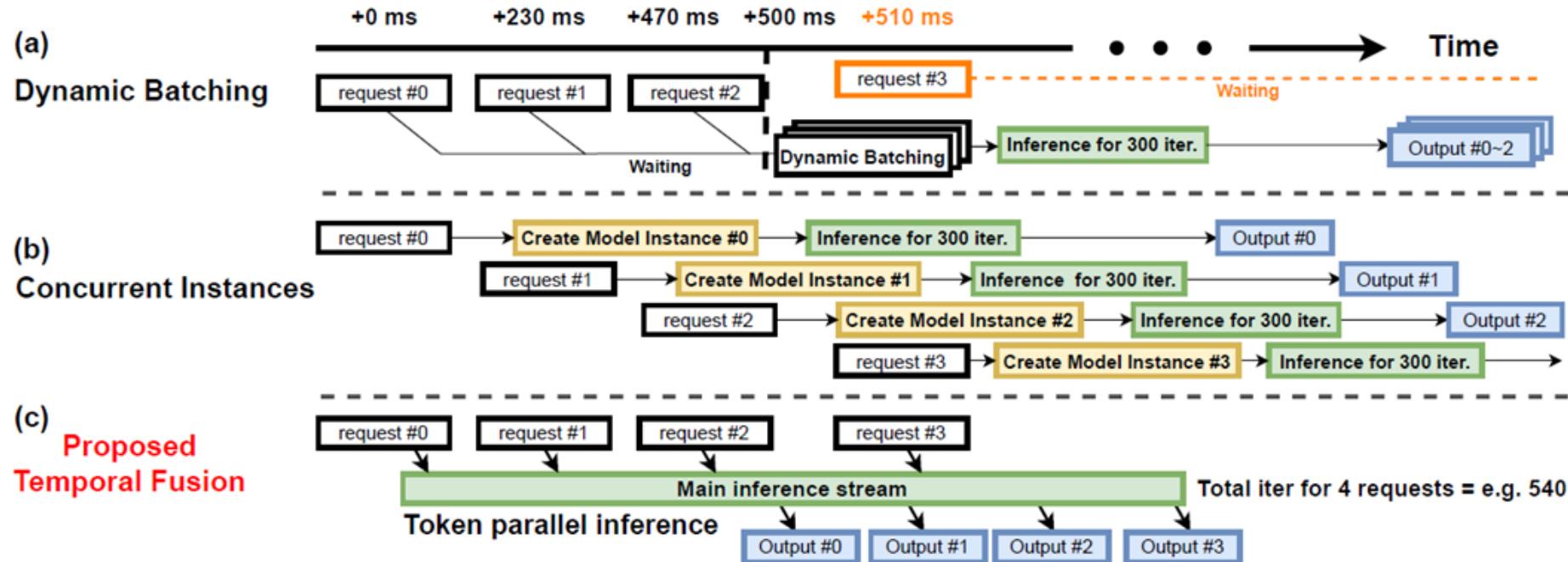
(c) Offline

Inference performance of OpenVINO and PyTorch using MLPerf Edge on the DenseNet-121 and VGG-19 models

[1]. Ahn, Hyunho, Tian Chen, Nawras Alnaasan, Aamir Shafi, Mustafa Abduljabbar, and Hari Subramoni. "Performance Characterization of using Quantization for DNN Inference on Edge Devices: Extended Version." 7th IEEE International Conference on Fog and Edge Computing

Flover: Efficient parallel inference on LLMs with temporal fusion^[1]

- When serving multiple requests, how to deliver both low-latency and high-throughput?
- For generative models such as GPT, LLaMA, the generation is sequential and regulated by 'for' loop.
 - For multiple requests that arrive at different time, how do we schedule the inference?

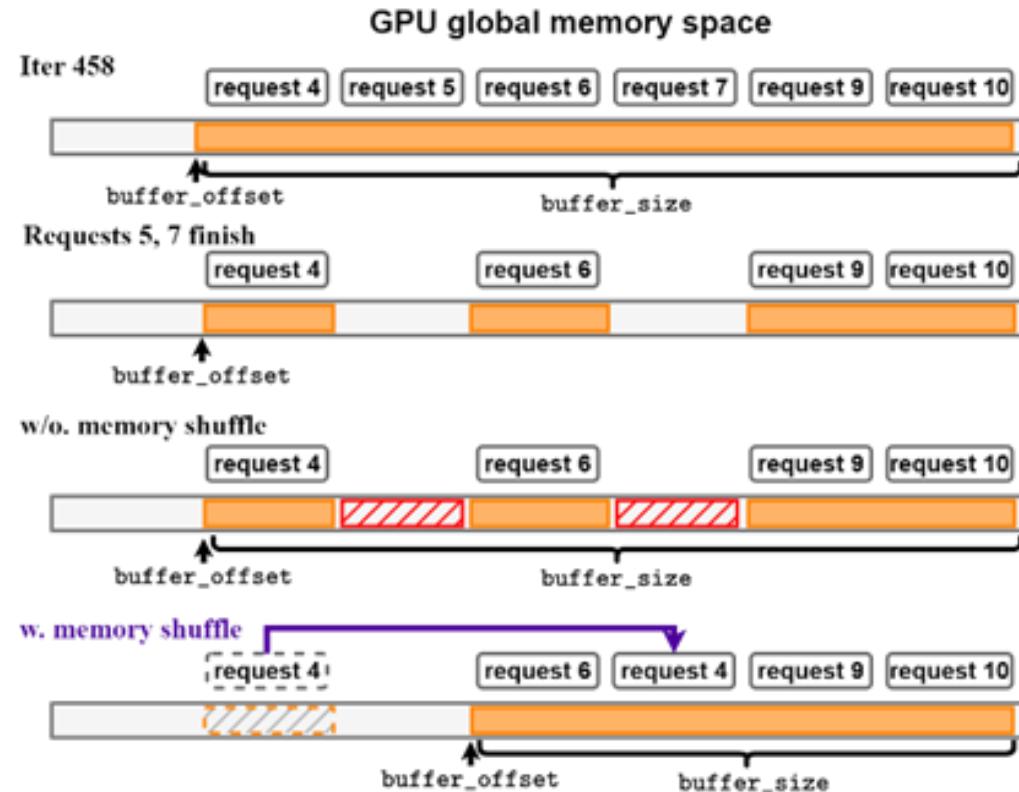


- We leverage the temporal property in generative model to smartly batch token generation.
 - Only maintain one persistent inference instance for serving any incoming requests with no delay.
 - Efficient memory reordering strategy to assure requests' buffer continuity, avoiding internal fragments.

[1] Yao, Jinghan, Nawras Alnaasan, Tian Chen, Aamir Shafi, and Hari Subramoni. "Flover: A Temporal Fusion Framework for Efficient Autoregressive Model Parallel Inference." In Proceeding of HiPC 23

Flover: Efficient parallel inference on LLMs with temporal fusion

- When requests evicted, their buffer need to be properly managed.
 - When early arrived requests finished
 - When request gets an EOS token



Memory Reordering

Outline

- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- **Open Issues and Challenges**
- Conclusion

Open Issues and Challenges

- Convergence of ML/DL and HPC
- ML/DL Benchmarks and Thoughts on Standardization
- Handling Trillion Parameter Models for Training and Inference
- Energy-aware and Fault-Tolerant DL training
- Low latency and high-throughput inference on a range of devices

Convergence of ML/DL and HPC

- Is Machine Learning/Deep Learning and Data Science an HPC Problem?
 - Distributed Model/DNN Training is definitely an HPC problem
 - Inference – not yet an HPC problem
 - Support for Machine Learning frameworks on HPC systems is improving (yet lagging)
- Why HPC can help?
 - Decades of research for communication models and performance optimizations
 - MPI, PGAS, and other communication runtimes can help for “data-parallel” training
- Some of the needs for ML/DL frameworks are an exact match
 - Compute intensive problem
- Some needs are new for distributed/parallel communication runtimes
 - Large Message Communication
 - CUDA-Aware Communication

ML/DL Benchmarks and Thoughts on Standardization

- Can we have a standardized interface?
 - Are we there yet?
 - Deep Learning Interface (DLI)? Inspired by Message Passing Interface (MPI)
 - What can be a good starting point?
 - Will it come from the HPC community or the DL community?
 - Can there be a collaboration across communities?
- What about standard benchmarks? Is there a need?
 - State-of-the-art
 - HKBU benchmarks - <http://dlbench.comp.hkbu.edu.hk>
 - Soumith Chintala's benchmarks - <https://github.com/soumith/convnet-benchmarks>
 - DAWN Bench – <https://dawn.cs.stanford.edu/benchmark/>
 - MLPerf – <https://www.mlperf.org> -- Latest and Widely Promoted now!

Handling Trillion Parameter Models for Training and Inference

- The community has crossed models with Billion Parameters
- Already thinking about Models with Trillion Parameters
 - Trillion Parameter Consortium (<https://www.anl.gov/cels/trillion-parameter-consortium>)
- Model Training and Inference with Trillion Parameters will require
 - Extremely Large-scale datacenters (~1 million GPUs)
 - Accelerators and/or Memory subsystems to hold the model during training and inference
 - Next-generation of architectures (CPUs, GPUs, Interconnects) and algorithms for training and inference

Energy-aware and Fault-Tolerant DL Training

- Training Models with Billion Parameters requires
 - Extremely-large data centers with hundred thousands of GPUs
 - Months of training time
- Consumes significant energy
- GPUs go through failures
- Significant focus on
 - New generation of hardware and software for reducing energy consumption
 - Newer Checkpointing and fault-tolerant schemes

Low latency and high-throughput inference on a range of devices

- Wide range of needs for inference
 - Multiple disciplines (engineering, medicine, agriculture, ...)
 - Range of edge devices (laptops, smart phones, drones, dedicated devices)
- Require inference schemes with
 - Low-latency
 - High-throughput
 - Reduced cost
- The inference workflow pipeline involving edge devices, network, and back-end servers need to be heavily optimized based on the needs

Outline

- Introduction
- Deep Learning Frameworks
- Deep Neural Network Training
- Distributed Data-Parallel Training
 - Lab 1: Hands-on Exercises (Data Parallelism)
- Latest Trends in High-Performance Computing Architectures
- Challenges in Exploiting HPC Technologies for DL
- Advanced Distributed Training
 - Lab 2: Hands-on Exercises (Advanced Parallelism)
- Distributed Inference Solutions
- Open Issues and Challenges
- **Conclusion**

Conclusion

- Exponential growth in Machine Learning/Deep Learning/Data Science frameworks
- Provided an overview of issues, challenges, and opportunities for designing efficient communication runtimes
 - Efficient, scalable, and hierarchical designs are crucial for ML/DL/Data Science frameworks
 - Co-design of communication runtimes and ML/DL/Data Science frameworks will be essential
- Worked on a set of hands-on exercises to demonstrate the complex interaction between DL/ML middleware with the underlying HPC technologies and middleware
- Need collaborative efforts to achieve the full potential

Funding Acknowledgments

Funding Support by



Equipment Support by



Acknowledgments to all the Heroes (Past/Current Students and Staffs)

Current Students (Under/Graduate)

- N. Alnaasan (Ph.D.)	- S. Gumaste (Ph.D.)	- J. Oswal (Ph.D.)	- S. Zhang (Ph.D.)
- Q. Anthony (Ph.D.)	- J. Hatef (Ph.D.)	- T. Tran (Ph.D.)	- S. Mohammad (M.S.)
- C.-C. Chen (Ph.D.)	- G. Kuncham (Ph.D.)	- L. Xu (P.h.D.)	- B. Lampe (B.S.)
- T. Chen (Ph.D.)	- S. Lee (Ph.D.)	- S. Xu (Ph.D.)	- N. Klein (B.S.)
- N. Contini (Ph.D.)	- B. Michalowicz (Ph.D.)	- J. Yao (Ph.D.)	

Current Research Specialist

- R. Motlagh

Current Software Engineers

- N. Shineman

- M. Lieber

Past Research Scientists

- K. Hamidouche

- S. Sur

- X. Lu

- M. Abduljabbar

- A. Shafi

Past Faculty

- H. Subramoni

Past Senior Research Associate

- J. Hashmi

Past Programmers

- A. Reifsteck

- D. Bureddy

- J. Perkins

- B. Seeds

- A. Guptha

- N. Pavuk

Past Research Specialist

- M. Arnold

- J. Smith

Past Students

- A. Awan (Ph.D.)	- T. Gangadharappa (M.S.)	- K. Kulkarni (M.S.)	- S. Pai (M.S.)	- S. Sur (Ph.D.)
- A. Augustine (M.S.)	- K. Gopalakrishnan (M.S.)	- R. Kumar (M.S.)	- S. Potluri (Ph.D.)	- K. K. Suresh (Ph.D.)
- P. Balaji (Ph.D.)	- R. Gulhane (M.S.)	- S. Krishnamoorthy (M.S.)	- J. Queiser (M.S.)	- K. Vaidyanathan
- M. Bayatpour (Ph.D.)	- J. Hashmi (Ph.D.)	- K. Kandalla (Ph.D.)	- K. Raj (M.S.)	(Ph.D.)
- R. Biswas (M.S.)	- M. Han (M.S.)	- M. Li (Ph.D.)	- R. Rajachandrasekar (Ph.D.)	- A. Vishnu (Ph.D.)
- S. Bhagvat (M.S.)	- W. Huang (Ph.D.)	- P. Lai (M.S.)	- B. Ramesh (Ph.D.)	- J. Wu (Ph.D.)
- A. Bhat (M.S.)	- A. Jain (Ph.D.)	- J. Liu (Ph.D.)	- D. Shankar (Ph.D.)	- W. Yu (Ph.D.)
- D. Buntinas (Ph.D.)	- J. Jani (M.S.)	- M. Luo (Ph.D.)	- G. Santhanaraman (Ph.D.)	- J. Zhang (Ph.D.)
- L. Chai (Ph.D.)	- W. Jiang (M.S.)	- A. Mamidala (Ph.D.)	- N. Sarkauskas (B.S. and M.S)	- Q. Zhou (Ph.D.)
- B. Chandrasekharan (M.S.)	- J. Jose (Ph.D.)	- G. Marsh (M.S.)	- V. Sathu (M.S.)	- N. Chmura (B.S.)
- S. Chakraborty (Ph.D.)	- M. Kedia (M.S.)	- V. Meshram (M.S.)	- N. Senthil Kumar (M.S.)	
- N. Dandapanthula (M.S.)	- K. S. Khorassani (Ph.D.)	- A. Moody (M.S.)	- A. Singh (Ph.D.)	
- V. Dhanraj (M.S.)	- S. Kini (M.S.)	- S. Naravula (Ph.D.)	- J. Sridhar (M.S.)	
- C.-H. Chu (Ph.D.)	- M. Koop (Ph.D.)	- R. Noronha (Ph.D.)	- S. Srivastava (M.S.)	
	- P. Kousha (Ph.D.)	- X. Ouyang (Ph.D.)	- H. Subramoni (Ph.D.)	

Past Post-Docs

- D. Banerjee	- H.-W. Jin	- E. Mancini	- A. Ruhela
- X. Besseron	- J. Lin	- K. Manian	- J. Vienne
- M. S. Ghazimirsaeed	- M. Luo	- S. Marcarelli	- H. Wang

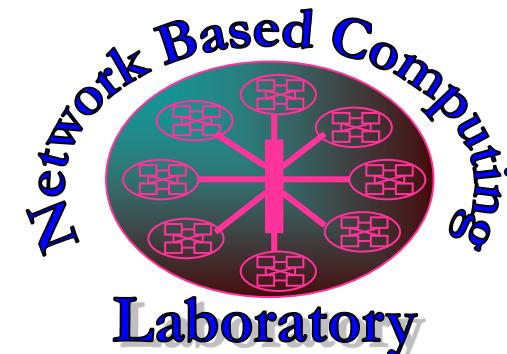
Thank You!

panda@cse.ohio-state.edu

alnaasan.1@osu.edu



<https://twitter.com/mvapich>



Network-Based Computing Laboratory

<http://nowlab.cse.ohio-state.edu/>



MVAPICH

MPI, PGAS and Hybrid MPI+PGAS Library

The MVAPICH2 Project

<http://mvapich.cse.ohio-state.edu/>



The High-Performance Deep Learning Project

<http://hidl.cse.ohio-state.edu/>