

# GPUs for ML

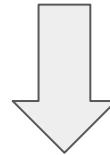
Serafina Di Gioia  
PostDoctoral Researcher @ ICTP

# What is ML?

in 1960

use and development of computer systems that are able to learn and adapt without following explicit instructions

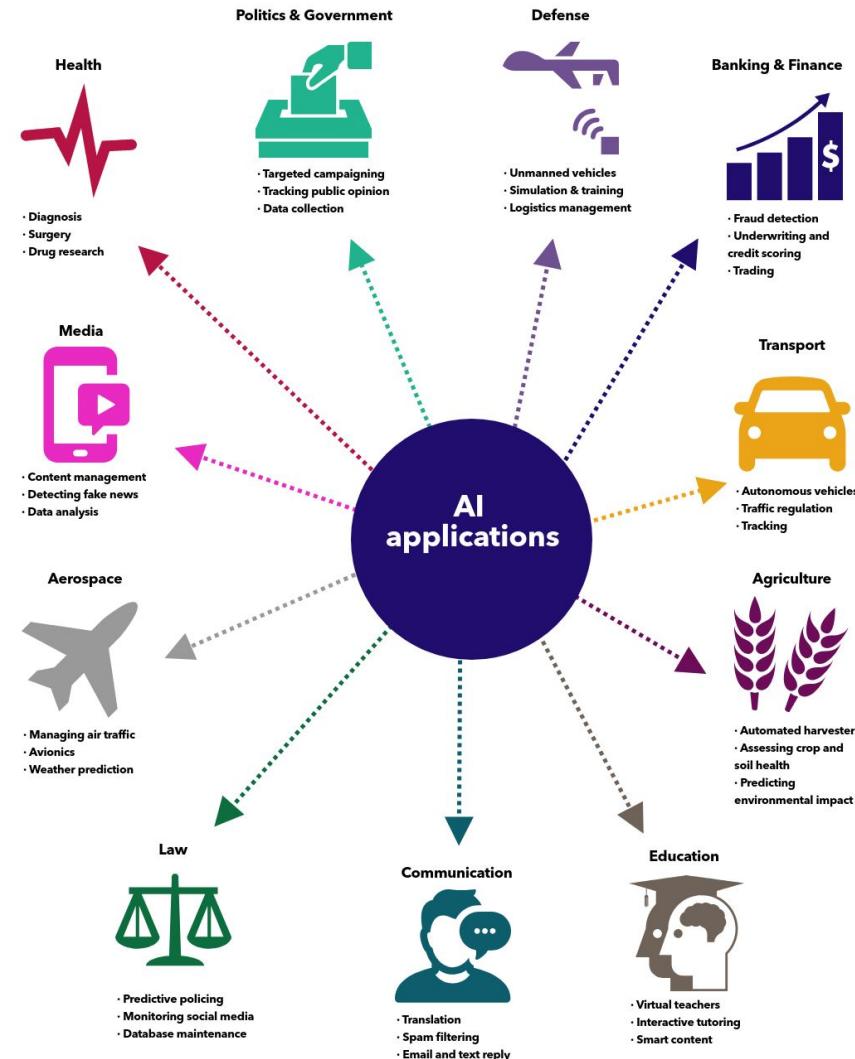
shift in perspective



in 2023

subfield of AI that uses algorithms trained on data to produce adaptable models that can perform a variety of complex tasks (e.g. classification and forecasting) without the need to code specific instructions for different tasks

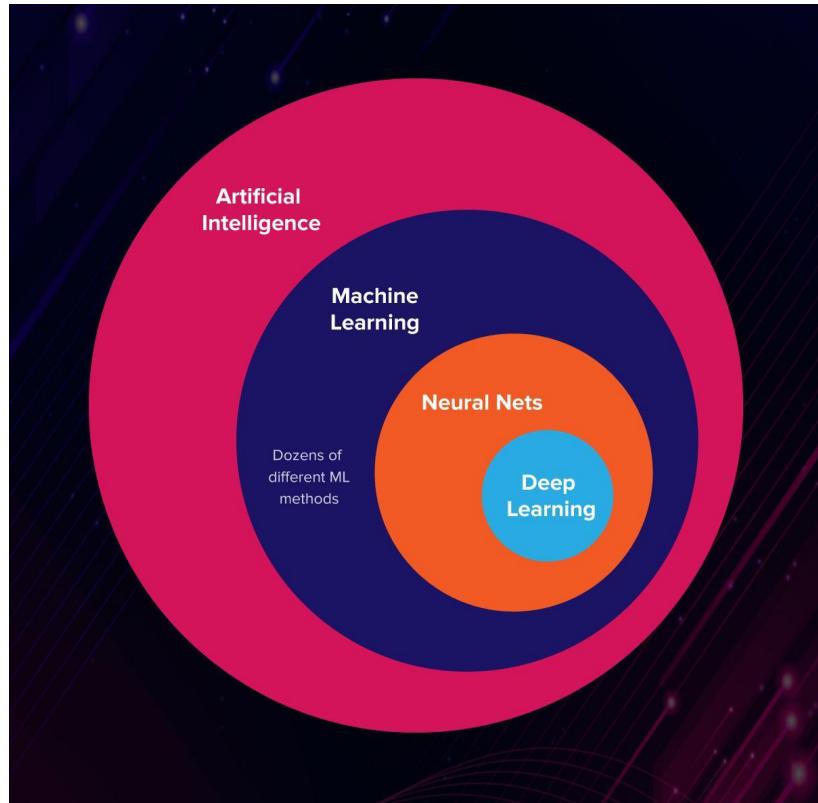
# What kind of task are solved by AI systems today?



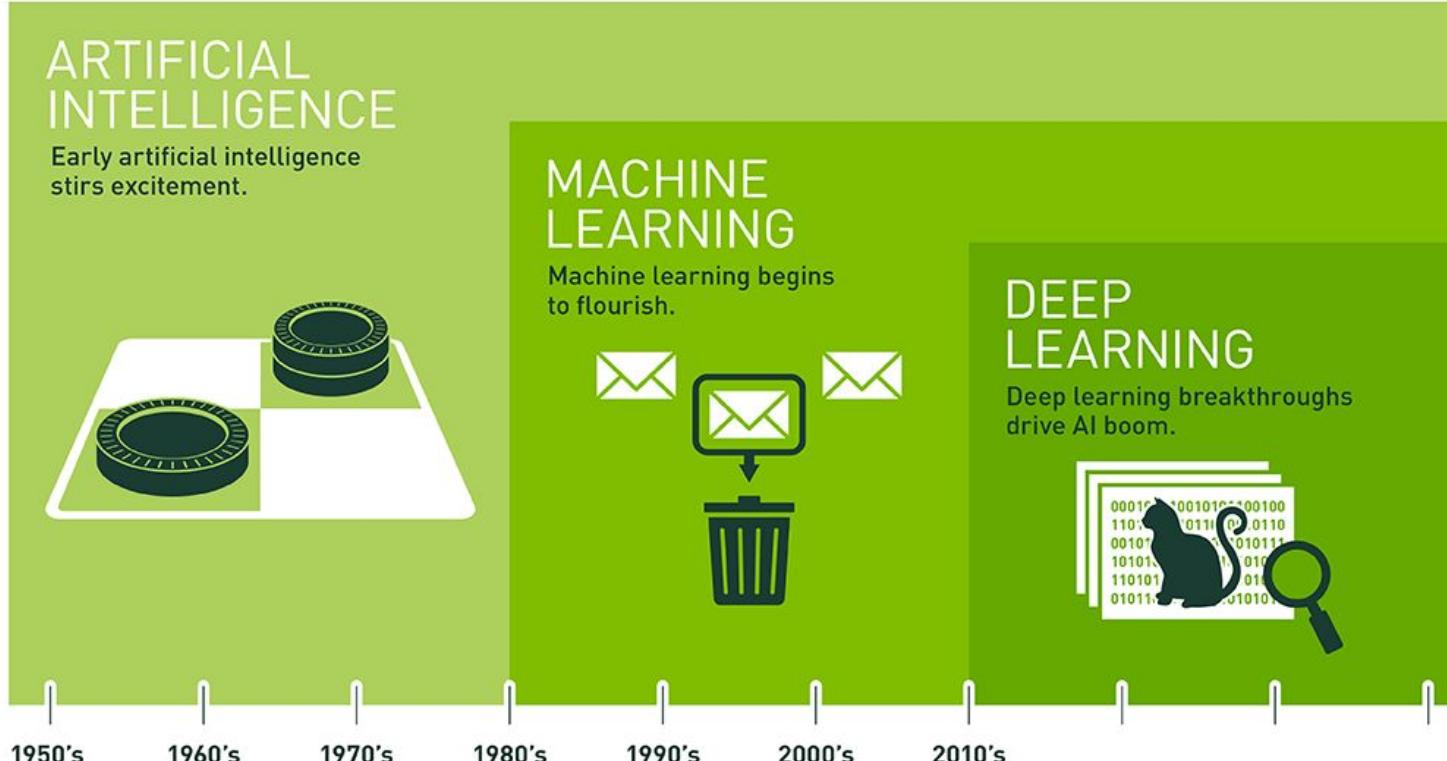
# Where is DL in the picture?

## Deep learning:

a type of machine learning based on artificial neural networks in which multiple layers of processing are used to extract progressively higher level features from data.



# From LISP to the DL revolution...

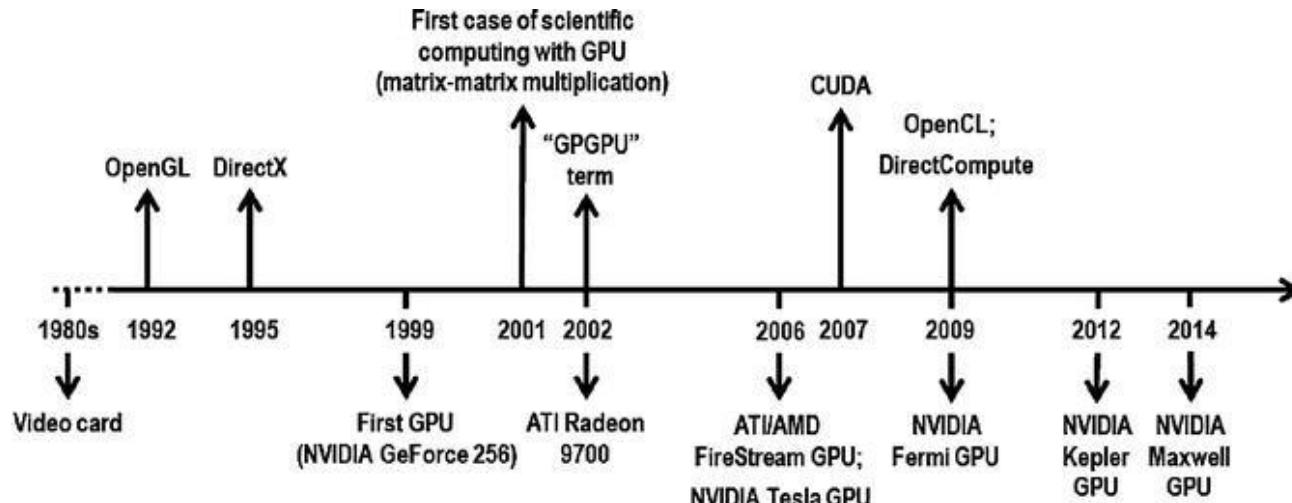


Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

# Main ingredients for DL breakthrough

- large datasets available (e.g IMAGENET)
- GPUs development (in particular, CUDA introduction)
- increased involvement of developers from CV and scientific communities

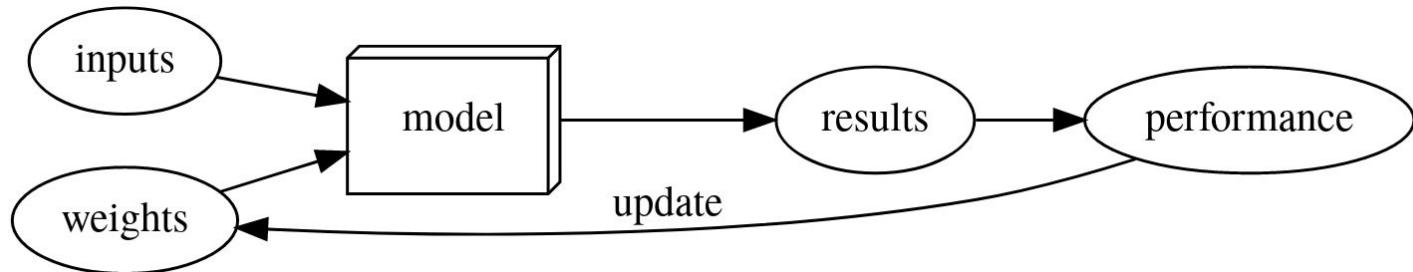
The DL era starts few years after that CUDA came to light



# Building blocks of ML algorithms

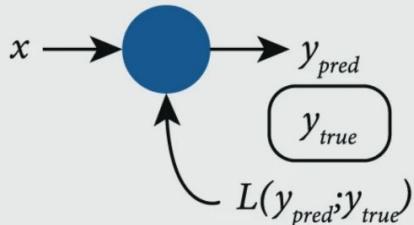
ML algorithms have three main components

1. **decision process:** based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data. This estimate can be used to solve a prediction or classification task
2. **error function:** it evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.
3. **Model Optimization Process:** If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this “evaluate and optimize” process, updating weights autonomously until a threshold of accuracy has been met.



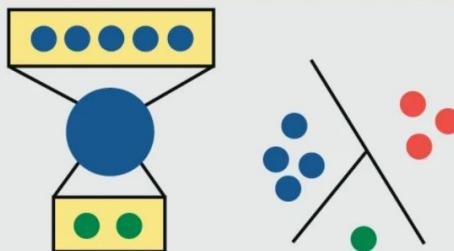
# ML scenarios

## Supervised learning



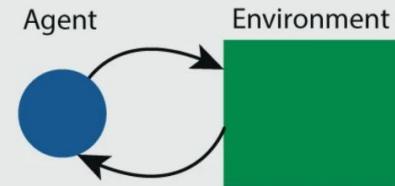
ML algorithm learns by comparing predicted and actual values

## Unsupervised learning



ML algorithm learns without labeled data (e.g. clustering, embedding)

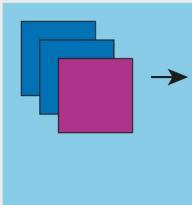
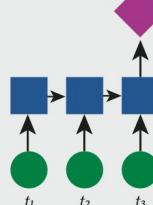
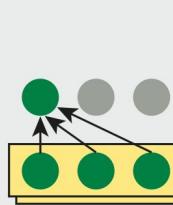
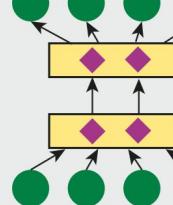
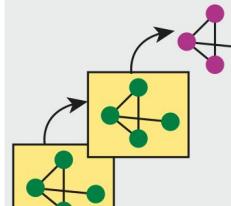
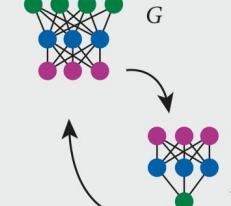
## Reinforcement learning



Agent (ML algorithm) learns by interacting with an environment

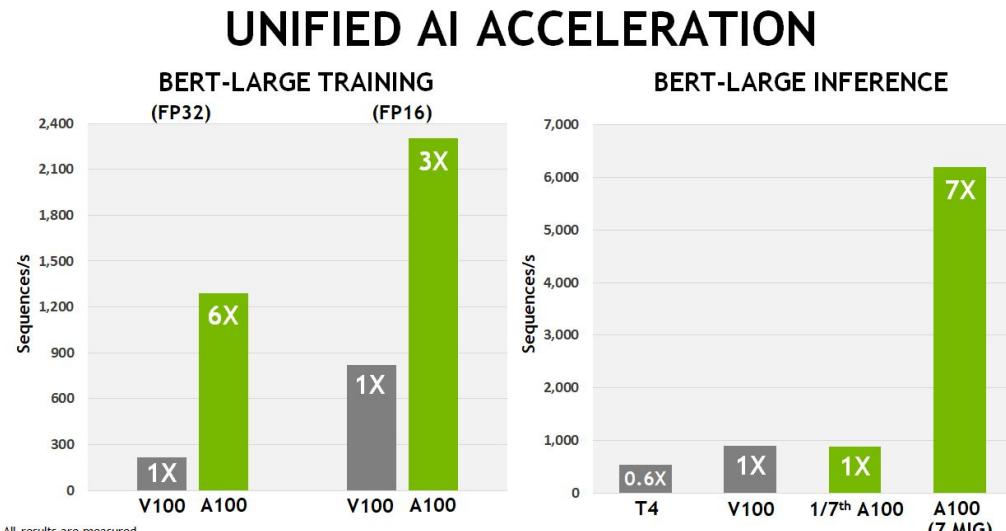
Learning type	Model building	Examples
Supervised	Algorithms or models learn from labeled data (task-driven approach)	Classification, regression
Unsupervised	Algorithms or models learn from unlabeled data (Data-Driven Approach)	Clustering, associations, dimensionality reduction
Semi-supervised	Models are built using combined data (labeled + unlabeled)	Classification, clustering
Reinforcement	Models are based on reward or penalty (environment-driven approach)	Classification, control

# Driving into the world of DL models

	Convolutional NN (CNN)	Recurrent NN (RNN)	Transformer	Autoencoder (AE)
Goal	Perform inference on data with local features	Perform inference on temporal data	Perform inference on sequential data	Embed high-dimensional data
Key idea	Learn shift-invariant filters	Learn temporal correlations via recurrent structure	Learn context based correlations via attention mechanism	Learn low-dimensional embedding of data
				
Goal	Graph NN (GNN)	Generative Adversarial Network (GAN)		
Key idea	Capture graph based dependencies in the data	Generate samples from data distribution		
			<p><b>Denoising autoencoders (DAE)</b> are autoencoder models that learn low dimensional embeddings of noisy high dimensional data, i.e. inputs that differ by a small amount of noise give rise to a similar embedding vector.</p> <p><b>Attention mechanism</b> mimics cognitive attention by learning importance weights for the inputs based on the whole input context (e.g. in a task of translating codons to amino acids attention mechanism will learn to give higher weight to the first two nucleic acids). Attention is the key part of transformer models, but can also be applied in conjunction with other layer types.</p> <p><b>Convolutional layers</b> have dimension which indicates the dimension of learned filters. Thus, we can have a 1-dimensional convolutional layer for sequences, 2-dimensional layer for matrices, and so on.</p> <p><b>Graph convolutional network (GCN)</b> is a graph neural network with convolutional layers defined by the topology of the graph. Thus instead of passing neighboring sequence or matrix entries through a filter, graph defined neighborhoods are used.</p>	

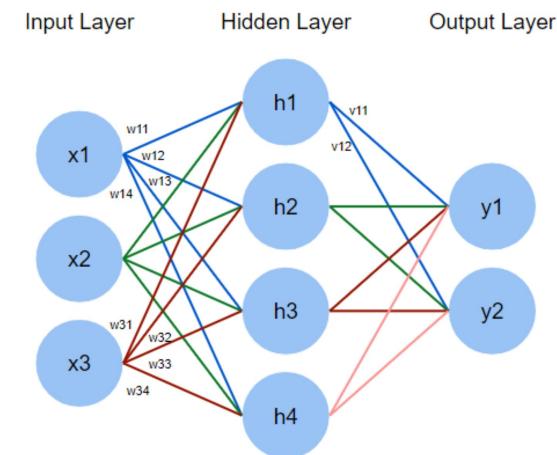
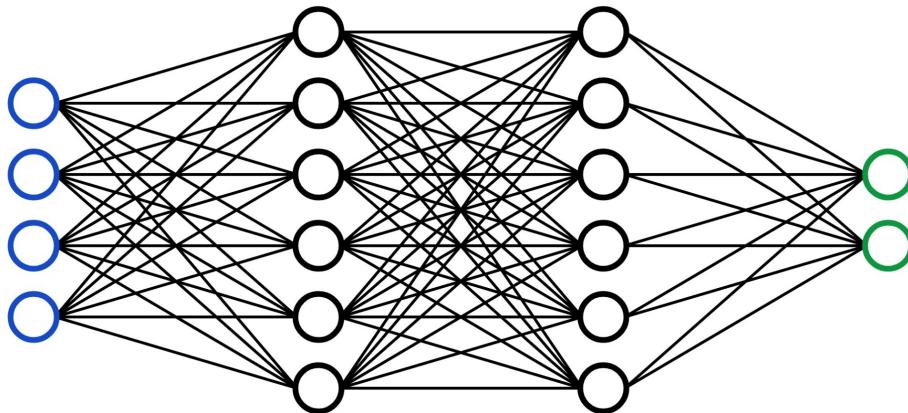
# Incredible performance increase of NVIDIA

	A100	V100	P100
FP32 CUDA Cores	6912	5120	3584
Boost Clock	~1.41GHz	1530MHz	1480MHz
Memory Clock	2.4Gbps HBM2	1.75Gbps HBM2	1.4Gbps HBM2
Memory Bus Width	5120-bit	4096-bit	4096-bit
Memory Bandwidth	1.6TB/sec	900GB/sec	720GB/sec
VRAM	40GB	16GB/32GB	16GB
Single Precision	19.5 TFLOPs	15.7 TFLOPs	10.6 TFLOPs
Double Precision	9.7 TFLOPs (1/2 FP32 rate)	7.8 TFLOPs (1/2 FP32 rate)	5.3 TFLOPs (1/2 FP32 rate)
INT8 Tensor	624 TOPs	N/A	N/A
FP16 Tensor	312 TFLOPs	125 TFLOPs	N/A
TF32 Tensor	156 TFLOPs	N/A	N/A
Interconnect	NVLink 3 12 Links (600GB/sec)	NVLink 2 6 Links (300GB/sec)	NVLink 1 4 Links (160GB/sec)
GPU	GA100 (826mm <sup>2</sup> )	GV100 (815mm <sup>2</sup> )	GP100 (610mm <sup>2</sup> )
Transistor Count	54.2B	21.1B	15.3B
TDP	400W	300W/350W	300W
Manufacturing Process	TSMC 7N	TSMC 12nm FFN	TSMC 16nm FinFET
Interface	SXM4	SXM2/SXM3	SXM
Architecture	Ampere	Volta	Pascal



All results are measured  
BERT Large Training (FP32 & FP16) measures Pre-Training phase, uses PyTorch including (2/3) Phase1 with Seq Len 128 and (1/3) Phase 2 with Seq Len 512,  
V100 is DGX1 Server with 8xV100, A100 is DGX A100 Server with 8xA100, A100 uses TF32 Tensor Core for FP32 training  
BERT Large Inference uses TRT 7.1 for T4/V100, with INT8/FP16 at batch size 256. Pre-production TRT for A100, uses batch size 94 and INT8 with sparsity

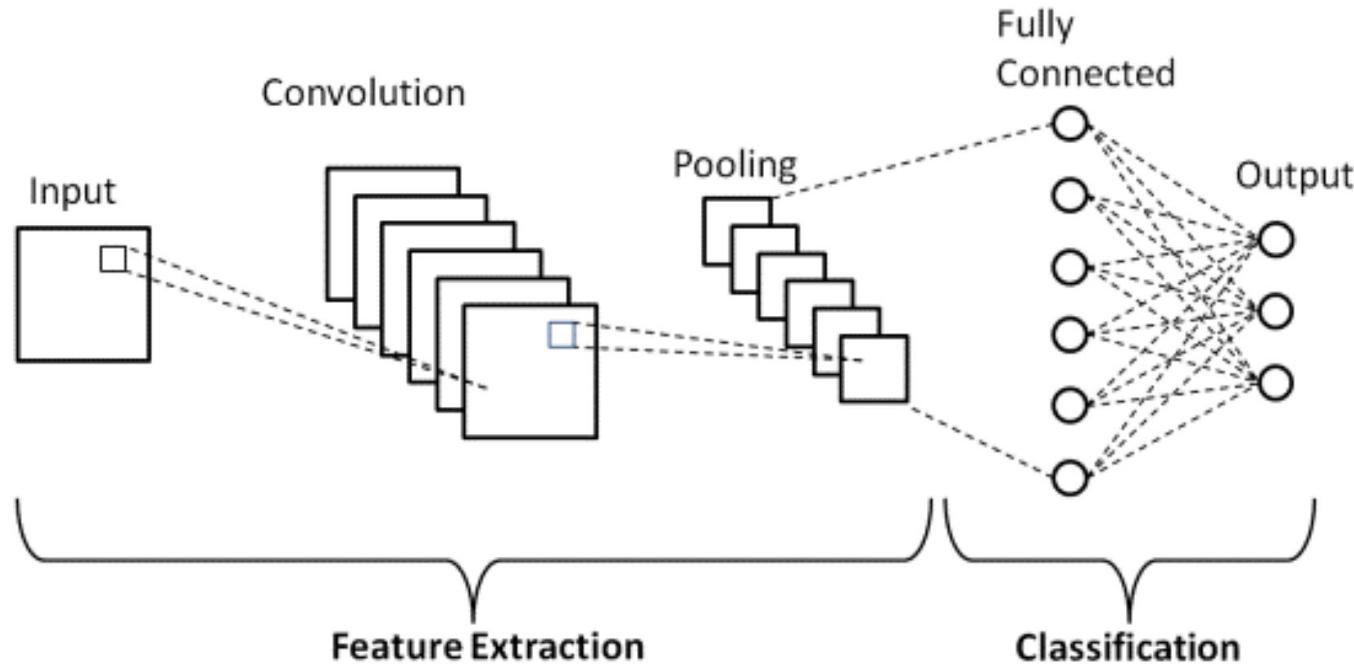
# Linear Neural Network (INN)



$$\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} * \begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \end{bmatrix} = \begin{bmatrix} h'_1 & h'_2 & h'_3 & h'_4 \end{bmatrix}$$

Alias of nn.linear() in Pytorch: torch.mm(inputs, linear.weight.T).add(linear.bias)

# DL for image classification: Convolutional Neural Networks (CNN)



# Everything in DL turns to be a matrix multiplication at the end...

Let's take a look at basic element of CNN: convolution layer

Consider the case where we are applying (2,2) kernel

$\alpha$	$\beta$
$\gamma$	$\delta$

to a (3,3) matrix:

A	B	C
D	E	F
G	H	J

$\alpha$	$\beta$
$\gamma$	$\delta$

applied to

$\alpha$	$\beta$
$\gamma$	$\delta$

A	B	C
D	E	F
G	H	J

yields

P

$\alpha$	$\beta$
$\gamma$	$\delta$

A	B	C
D	E	F
G	H	J

Q

$\alpha$	$\beta$
$\gamma$	$\delta$

A	B	C
D	E	F
G	H	J

R

$\alpha$	$\beta$
$\gamma$	$\delta$

A	B	C
D	E	F
G	H	J

S

The convolution can be rewritten as

$$\begin{array}{c}
 \begin{array}{ccccccccc}
 \alpha & \beta & 0 & y & \delta & 0 & 0 & 0 & 0 \\
 0 & \alpha & \beta & 0 & y & \delta & 0 & 0 & 0 \\
 0 & 0 & 0 & \alpha & \beta & 0 & y & \delta & 0 \\
 0 & 0 & 0 & 0 & \alpha & \beta & 0 & y & \delta
 \end{array} * \begin{array}{c}
 A \\
 B \\
 C \\
 D \\
 E \\
 F \\
 G \\
 H \\
 J
 \end{array} + \begin{array}{c}
 b \\
 b \\
 b \\
 b \\
 b
 \end{array} = \begin{array}{l}
 \alpha A + \beta B + 0C + yD + \delta E + 0F + 0G + 0H + 0J + b \\
 0A + \alpha B + \beta C + 0D + yE + \delta F + 0G + 0H + 0J + b \\
 0A + 0B + 0C + \alpha D + \beta E + 0F + yG + \delta H + 0J + b \\
 0A + 0B + 0C + 0D + \alpha E + \beta F + 0G + yH + \delta J + b
 \end{array} = \begin{array}{l}
 \alpha A + \beta B + yD + \delta E + b \\
 \alpha B + \beta C + yE + \delta F + b \\
 \alpha D + \beta E + yG + \delta H + b \\
 \alpha E + \beta F + yH + \delta J + b
 \end{array} = \begin{array}{c}
 P \\
 Q \\
 R \\
 S
 \end{array}
 \end{array}$$

A B C D E F G H J

# Why GPUs for DL?

- 1) Neural networks are embarrassingly parallel algorithm
- 2) most of the operations performed in DL models can be rewritten as matrix multiplications
- 3) big datasets require to perform big matrix computation (extremely slow on CPU with respect to GPU)
- 4) well established libraries, with specific classes for ML objects (e.g. cuDNN, more recently tensorRT)

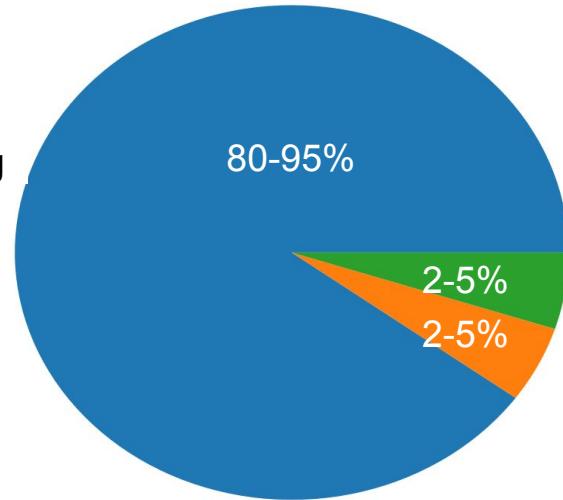
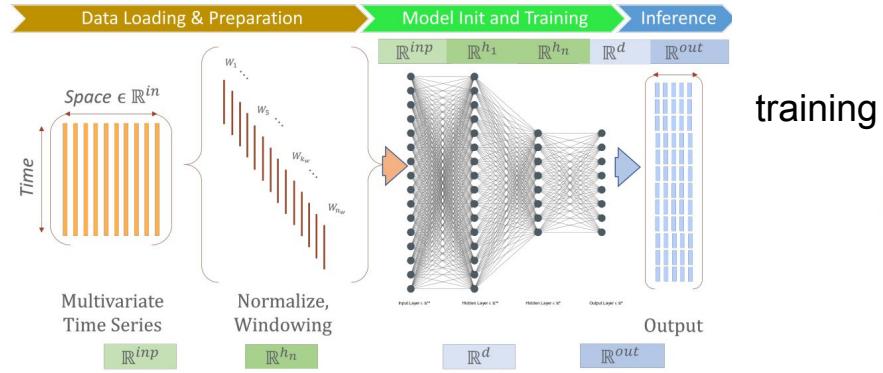
and we know that GPUs are very good in solving specific parallel tasks (e.g matrix multiplication), thanks to

- +1000 cores (>100K threads)
- SIMD / SIMT
- high memory bandwidth
- newer GPUs have also tensor cores (particularly suited to tensor ops typical of NNs), and mixed precision

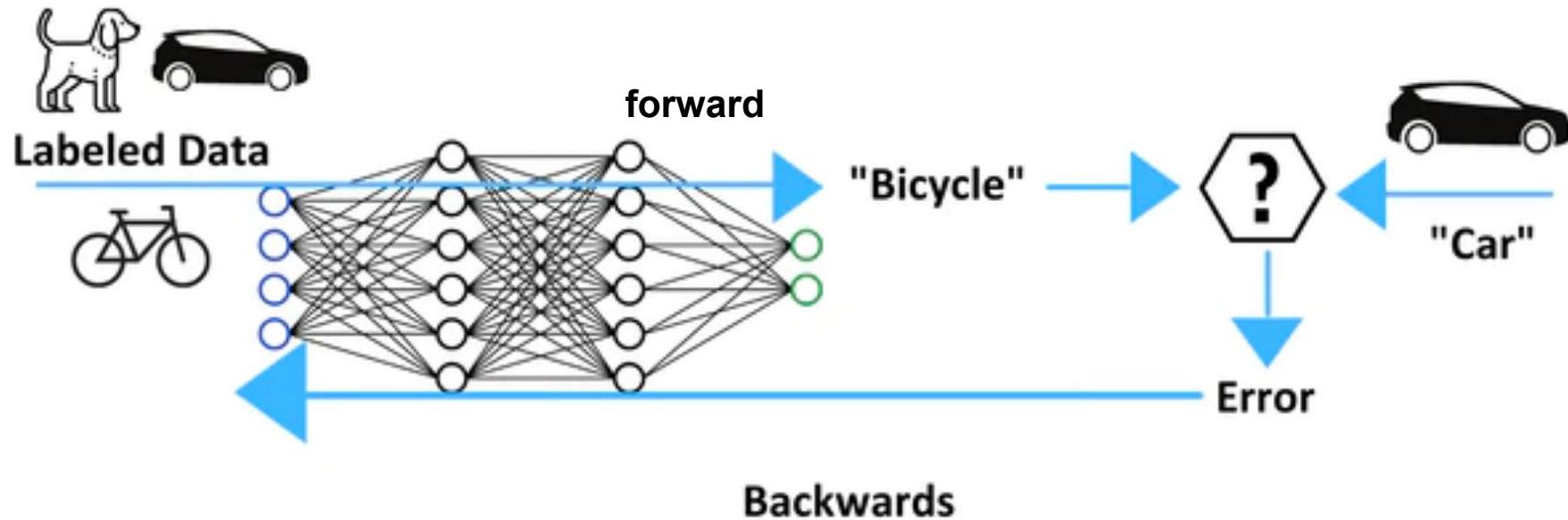
However, also GPUs have limitations:

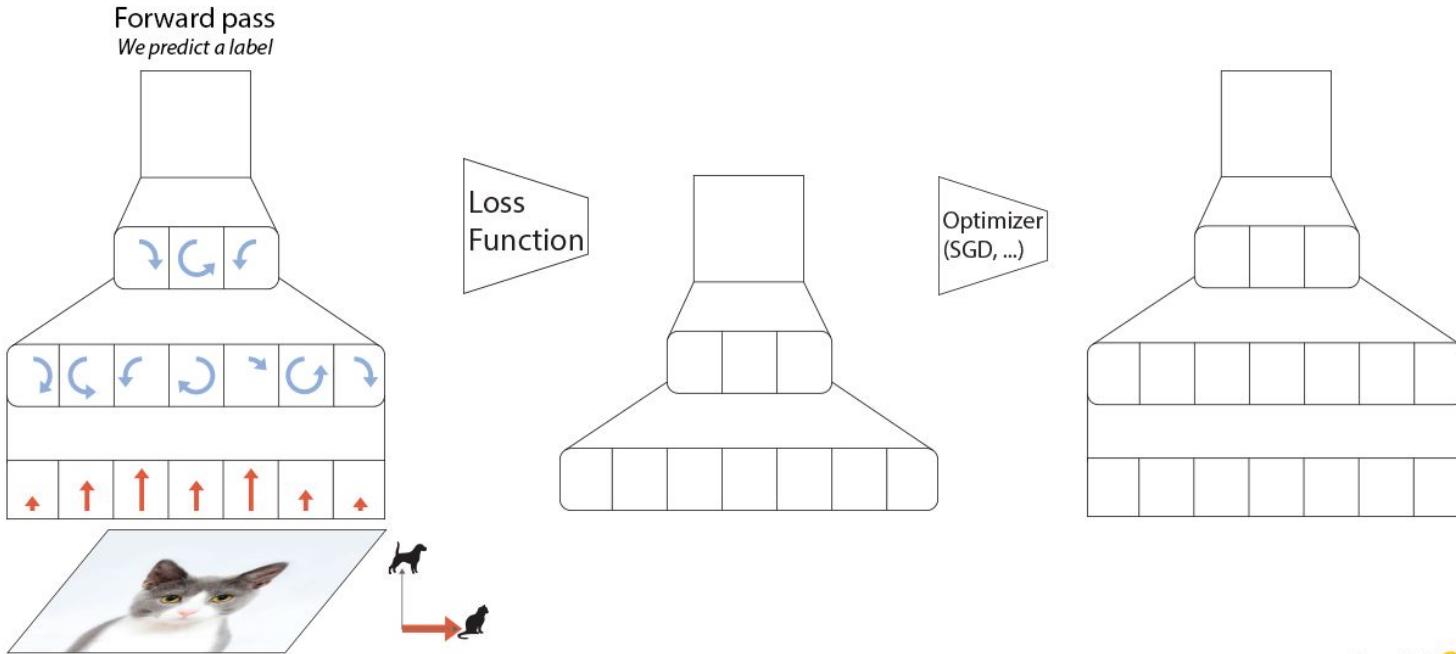
- GPUs might not be as efficient for extreme sparse networks, due to the overhead of managing sparse data structures.
- Some specialized sparse operations might not be as optimized as dense operations on GPUs.

# Analyzing the computational workload of DL models



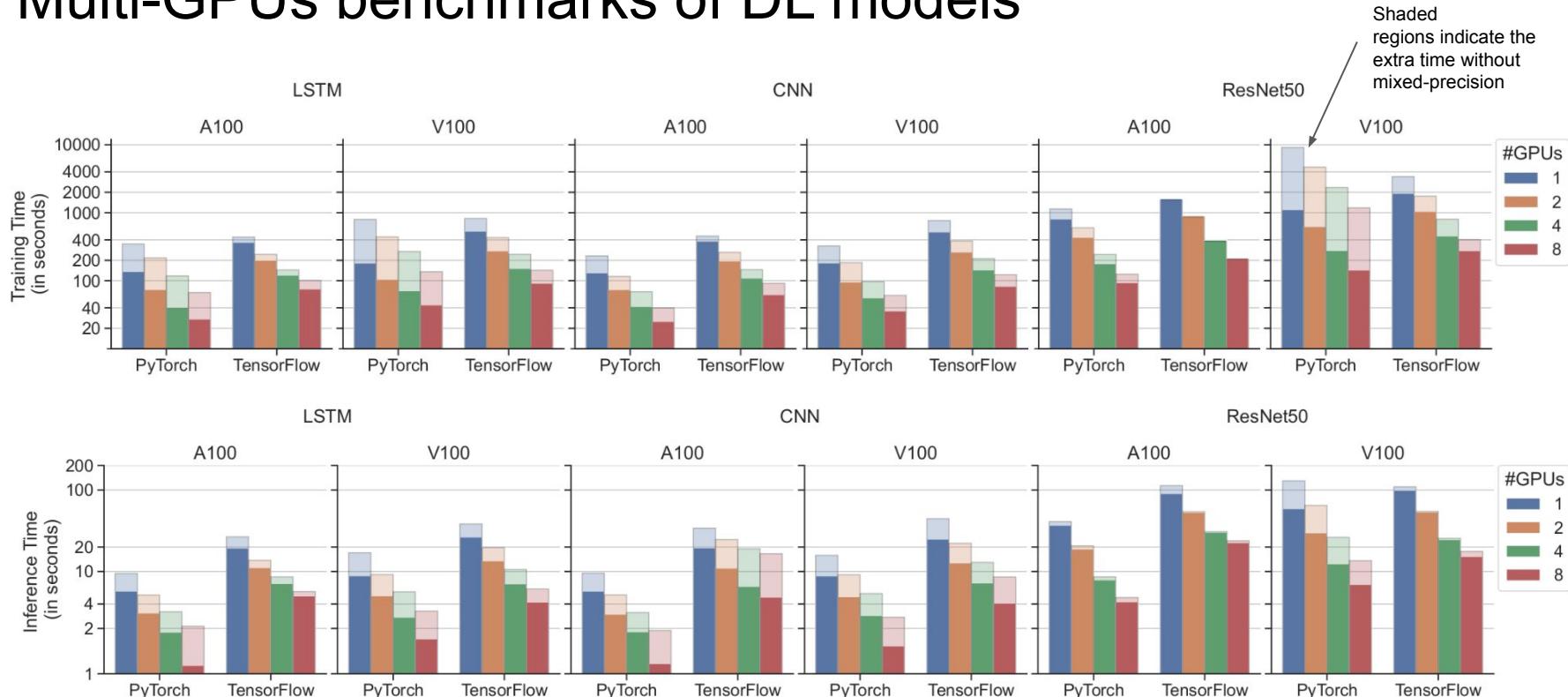
# Why training is so much compute-intensive?





@Thom\_Wolf 😊

# Multi-GPUs benchmarks of DL models



# Different strategies for Multi-GPUs training

we can identify 5 different categories of parallelism

tensor parallelism

model parallelism

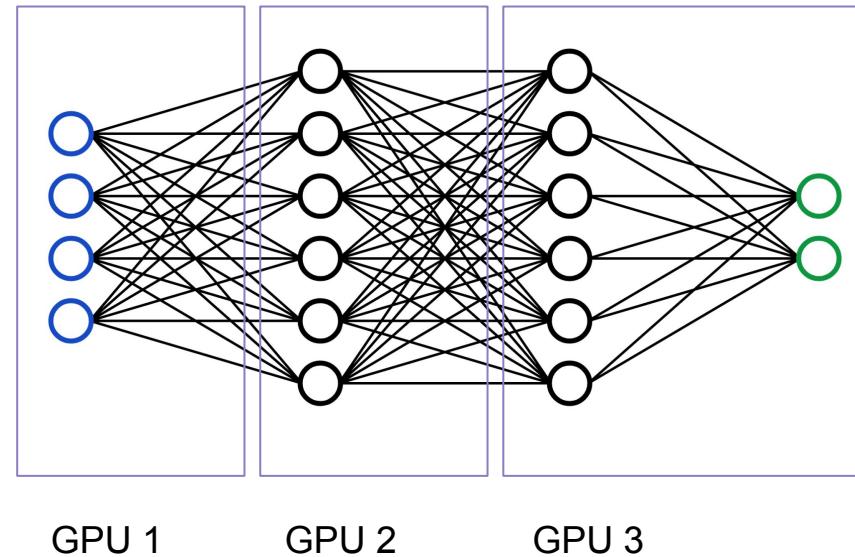
data parallelism

sequence parallelism

pipeline parallelism

# Model parallelism

In this parallelism framework we choose to put different layers of the NN on different GPUs to work around GPU memory limits

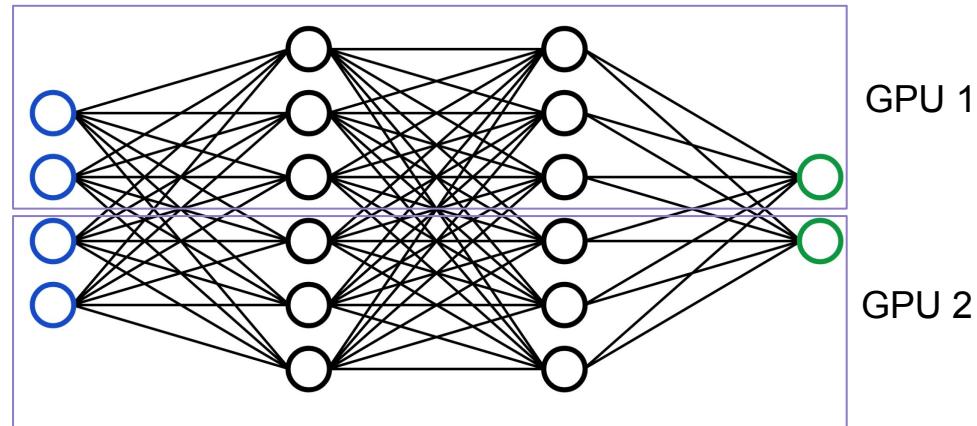


# Tensor parallelism

In this framework we split the tensor operation done at each layer among different GPUs

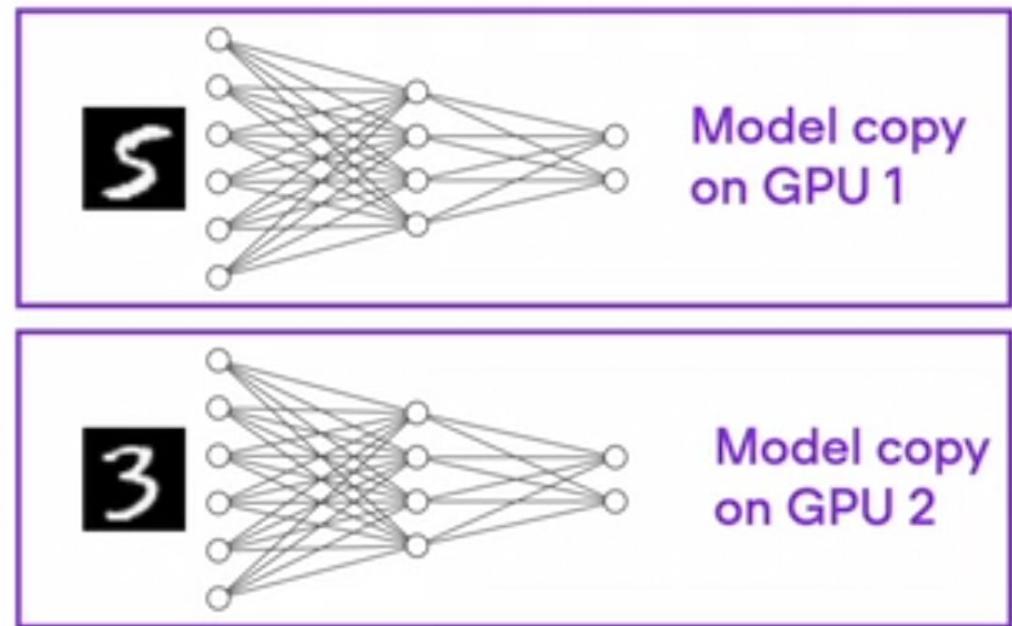
similarly to what we would have done for matmul

$$\begin{array}{c} \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 22 & 28 \\ \hline 49 & 64 \\ \hline \end{array} \end{array} \quad \left| \quad \begin{array}{c} \text{GPU 1} \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 3 & 4 & 5 \\ \hline 5 & 6 & 6 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 22 & 28 \\ \hline 49 & 64 \\ \hline \end{array} \end{array} \right. \quad \begin{array}{c} \text{GPU 2} \\ \begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 22 & 28 \\ \hline 49 & 64 \\ \hline \end{array} \end{array}$$



# Data Parallelism

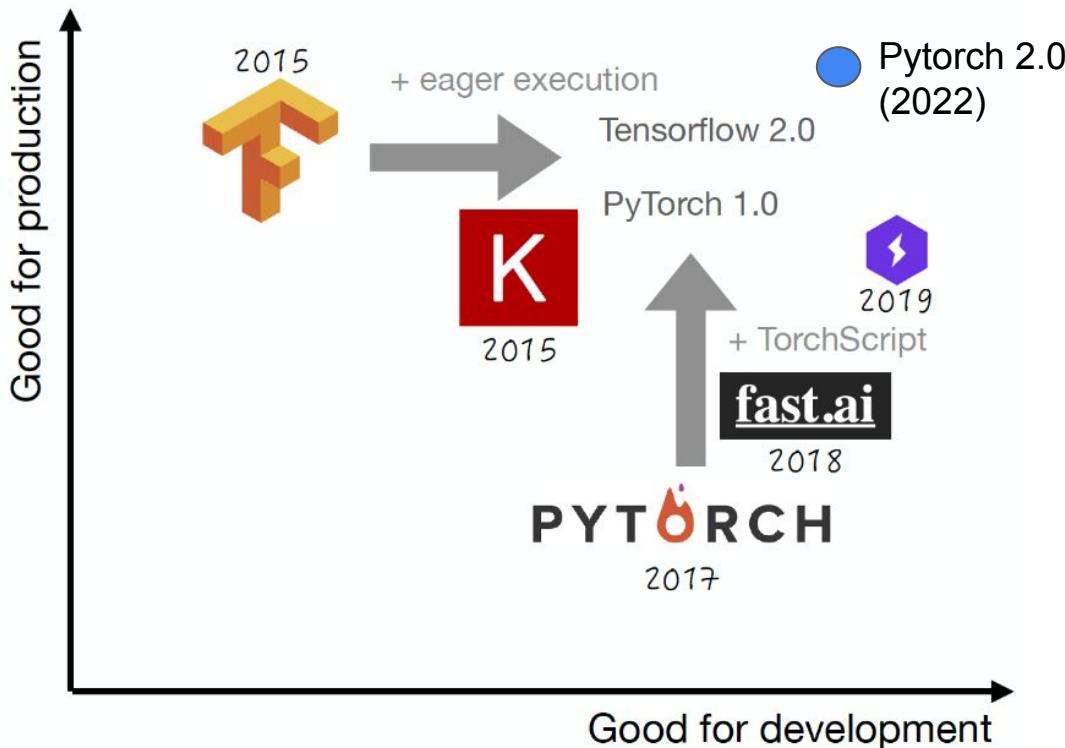
In this framework we split batches to train DL model into different GPUs



# More complex strategies for DL training

Sequence parallelism and pipeline parallelism frameworks are obtained combining the previous approaches, and are typically applied to DL models dealing with spatio-temporal data.

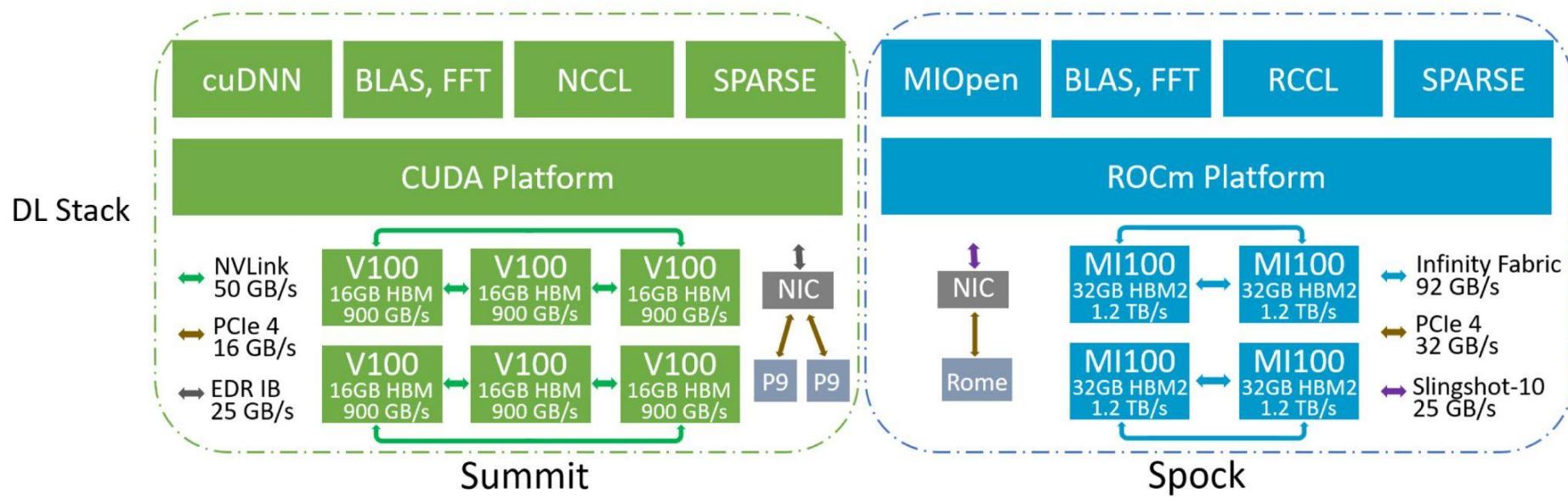
# Python most used libraries/frameworks for DL



# What's behind Pytorch/Tensorflow?

Framework

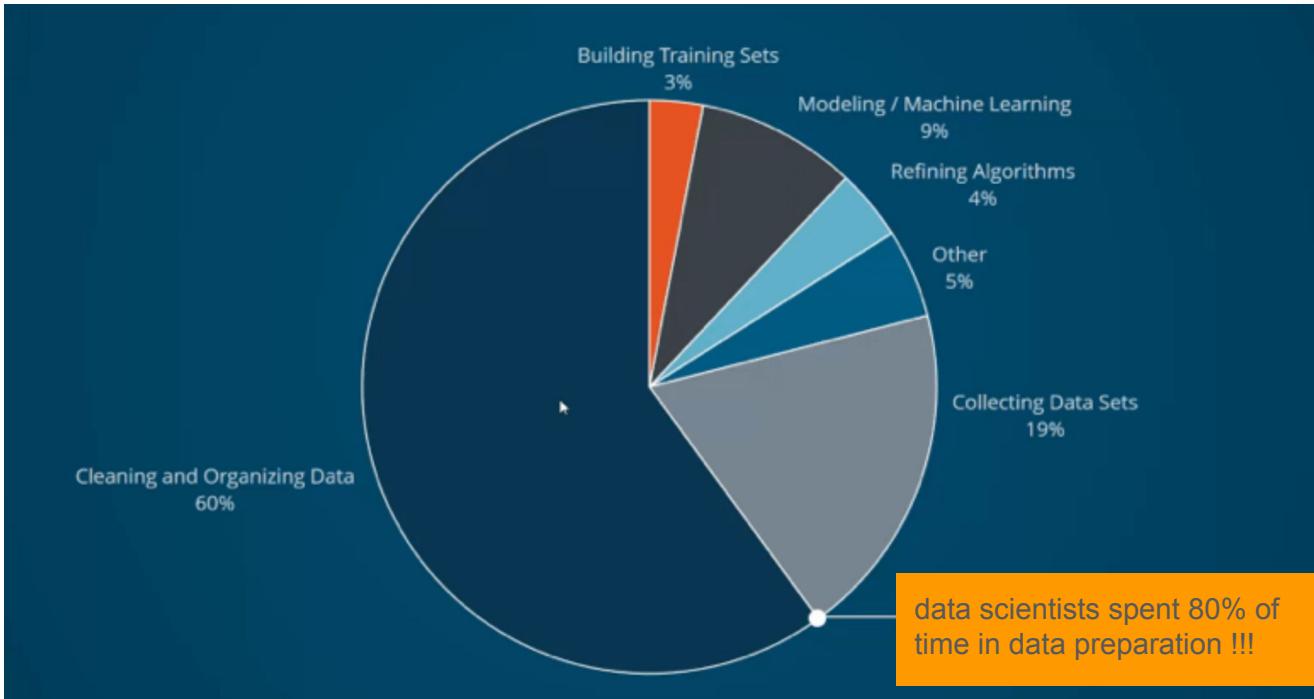
TensorFlow, PyTorch, Caffe, MxNet



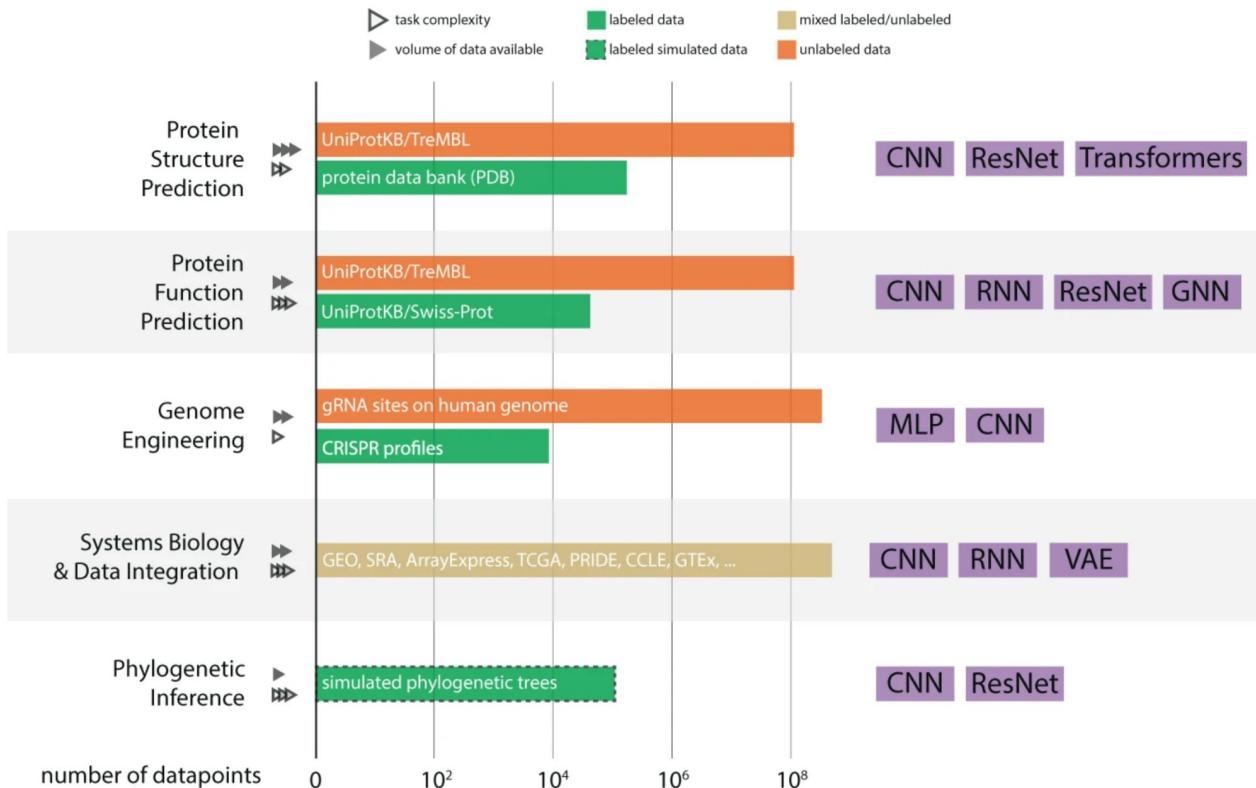
# Latest NVIDIA developments (HW/SW)

	DESKTOP	DATACENTER AND CLOUD
TRAINING AND INFERENCE	 DGX Station	 DGX-2  DGX-1  Tesla V100
AUTONOMOUS MACHINES	 Jetson TX2	 AI SELF-DRIVING PLATFORM
NVIDIA DEEP LEARNING SDK and CUDA		

# Do we need GPUs also for other ML tasks?

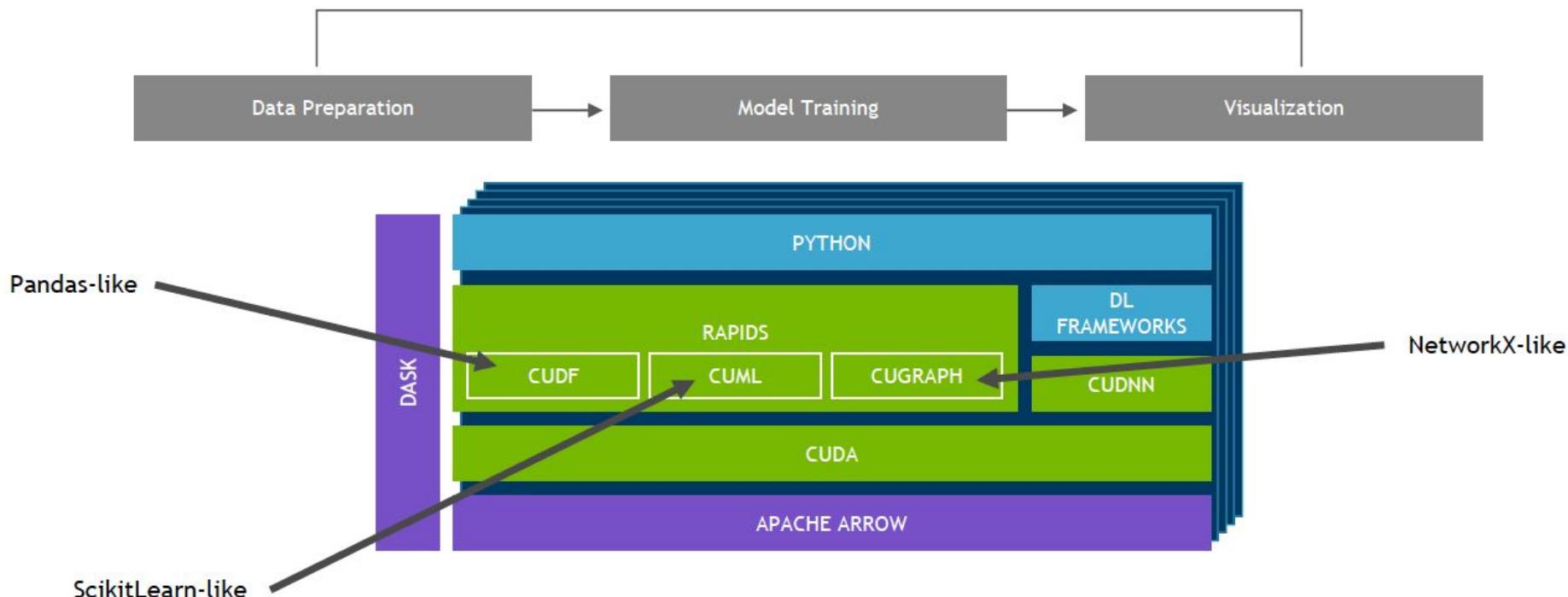


# Typical sizes of DL data sets

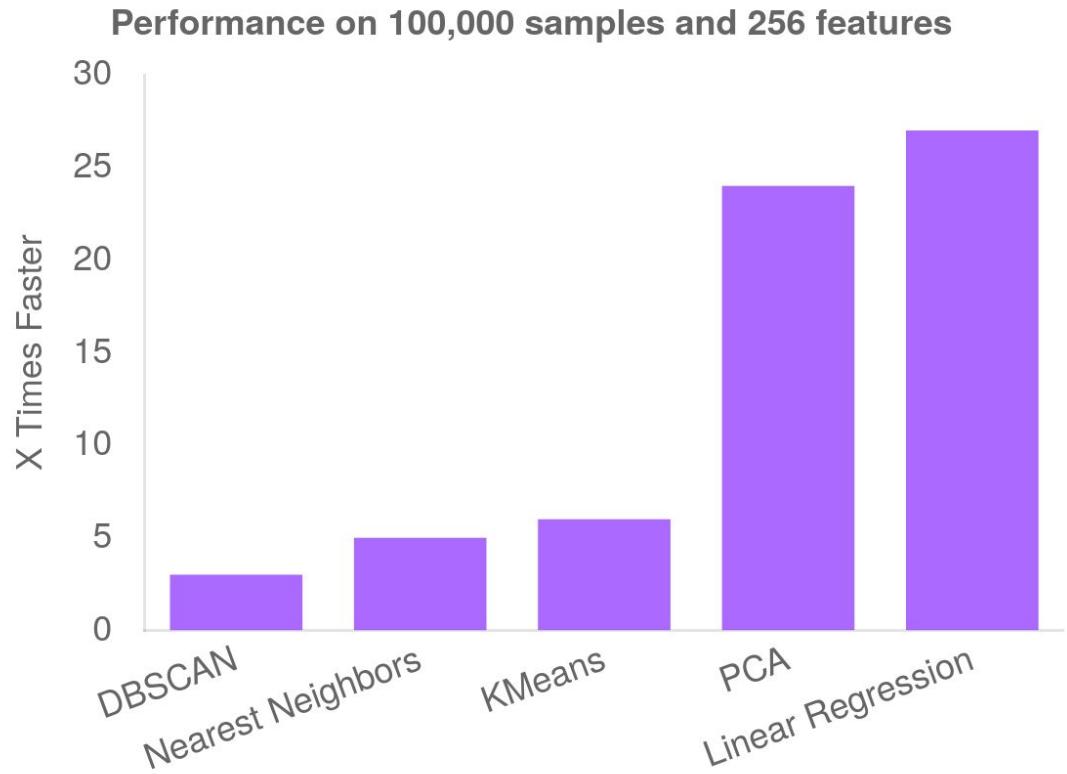


The increasing complexity of the new datasets, typical of big data epoch motivates the need for GPU-based libraries for feature analysis and data preprocessing

# GPU-based libraries outside of Pytorch

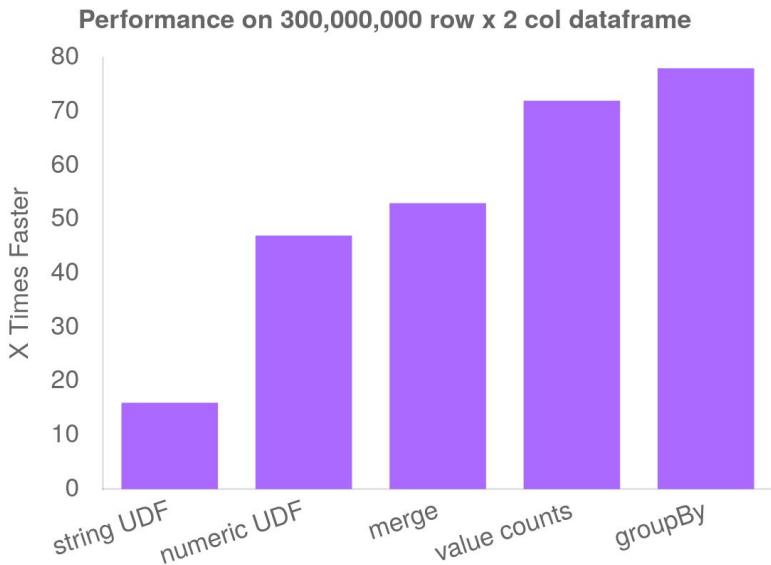


# GPU for classical ML



\* Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/  
512GB and NVIDIA A100 80GB (1x GPU) w/ scikit-learn v1.2 and  
cuML v23.02

# GPUs for data preprocessing



\* Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/  
512GB and NVIDIA A100 80GB (1x GPU) w/ pandas v1.5 and cuDF  
v23.02

# References

Milan Jain, Sayan Ghosh, Sai Pushpak Nandanoori, 2022, *Workload Characterization of a Time-Series Prediction System for Spatio-Temporal Data*

Junqi Yin et. al, 2021, *Comparative evaluation of deep learning workloads for leadership-class systems*

NVIDIA Booklet on GPU development, 2021

*Deep Learning: A Comprehensive Overview on Techniques, Taxonomy, Applications and Research Directions*