AI For Science At Scale – Introduction

Sajal Dash, Junqi Yin, Wes Brewer

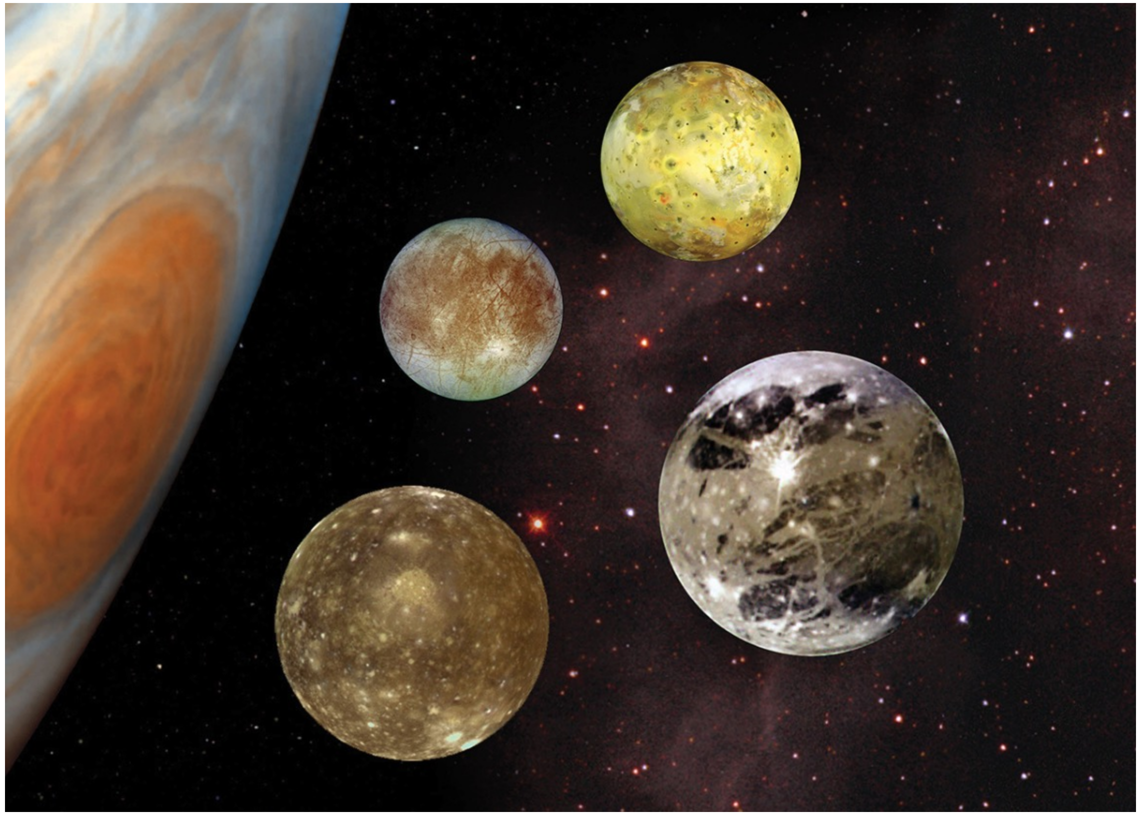
dashs@ornl.gov

Outline

- Introduction to statistical learning (linear regression)
- Intro to deep learning and training methods
- Short survey on scientific applications of deep learning
- Solving Spacegroup Classification

Learning from Data

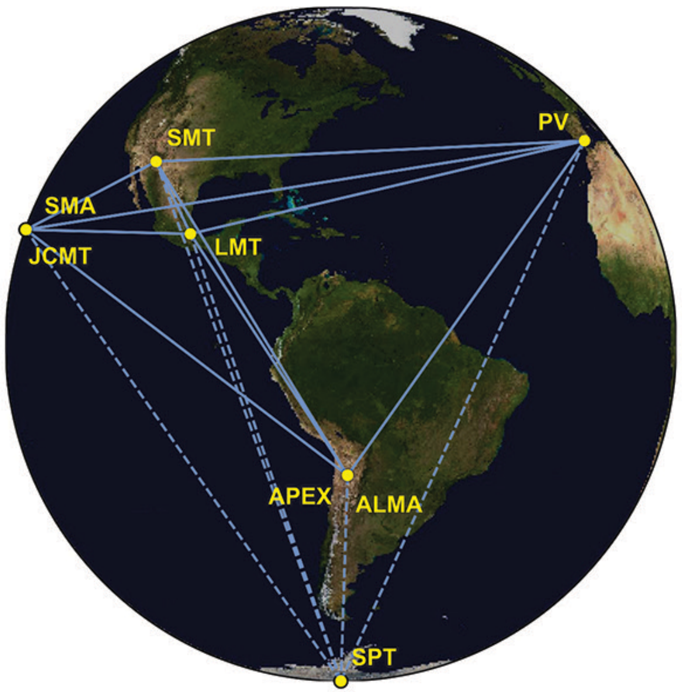
Galileo with his Telescope

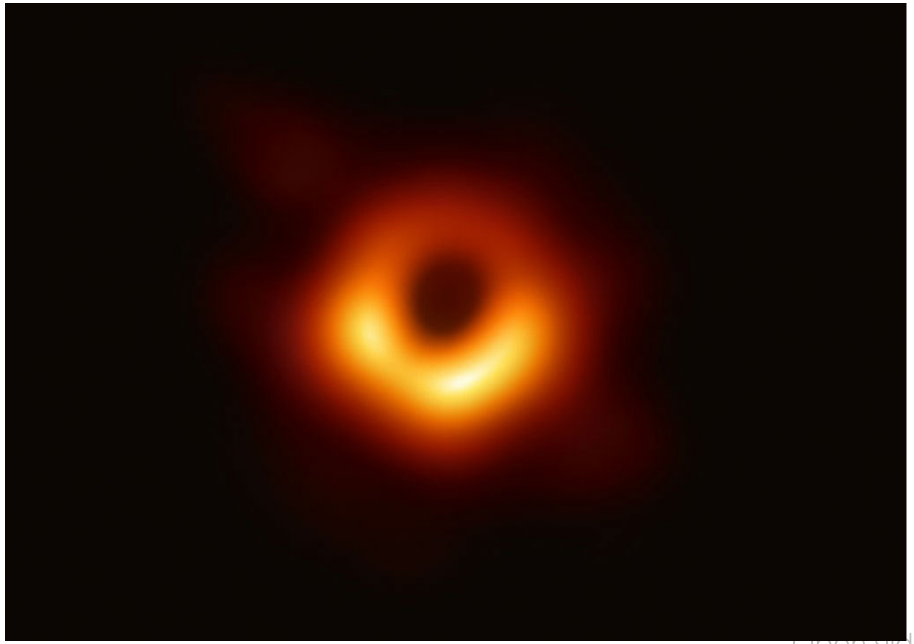


This "family portrait," a composite of the Jovian system, includes the edge of Jupiter with its Great Red Spot, and Jupiter's four largest moons, known as the Galilean satellites. From top to bottom, the moons shown are Io, Europa, Ganymede, and Callisto. Credit: NASA/JPL/DLR

Big Data Requires Automated Analysis

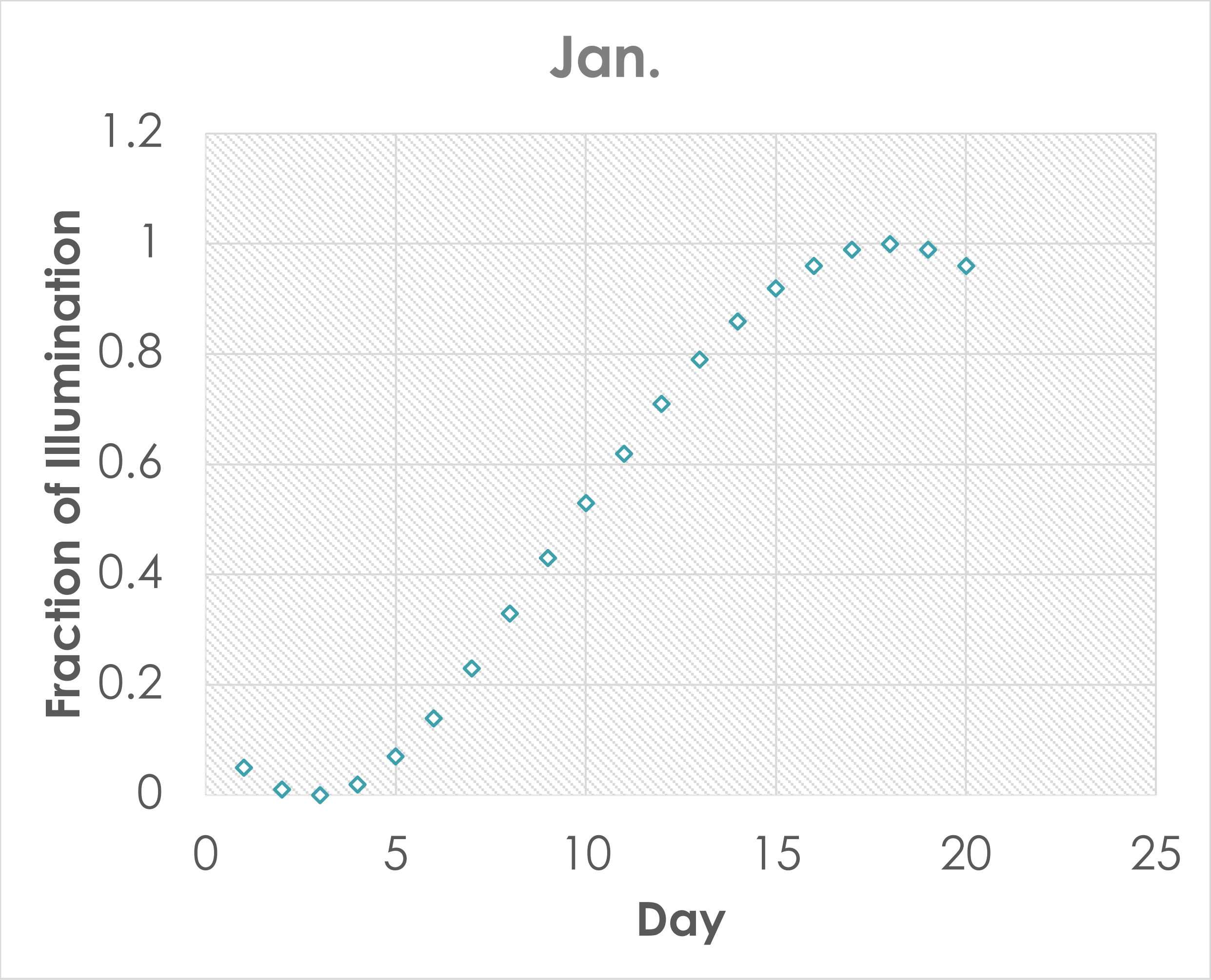
- In the last century, you could publish your observation data as a part of your manuscript and analyze them using pen and paper.
- Recent projects might generate Petabytes of data that needs to shipped across the globe.





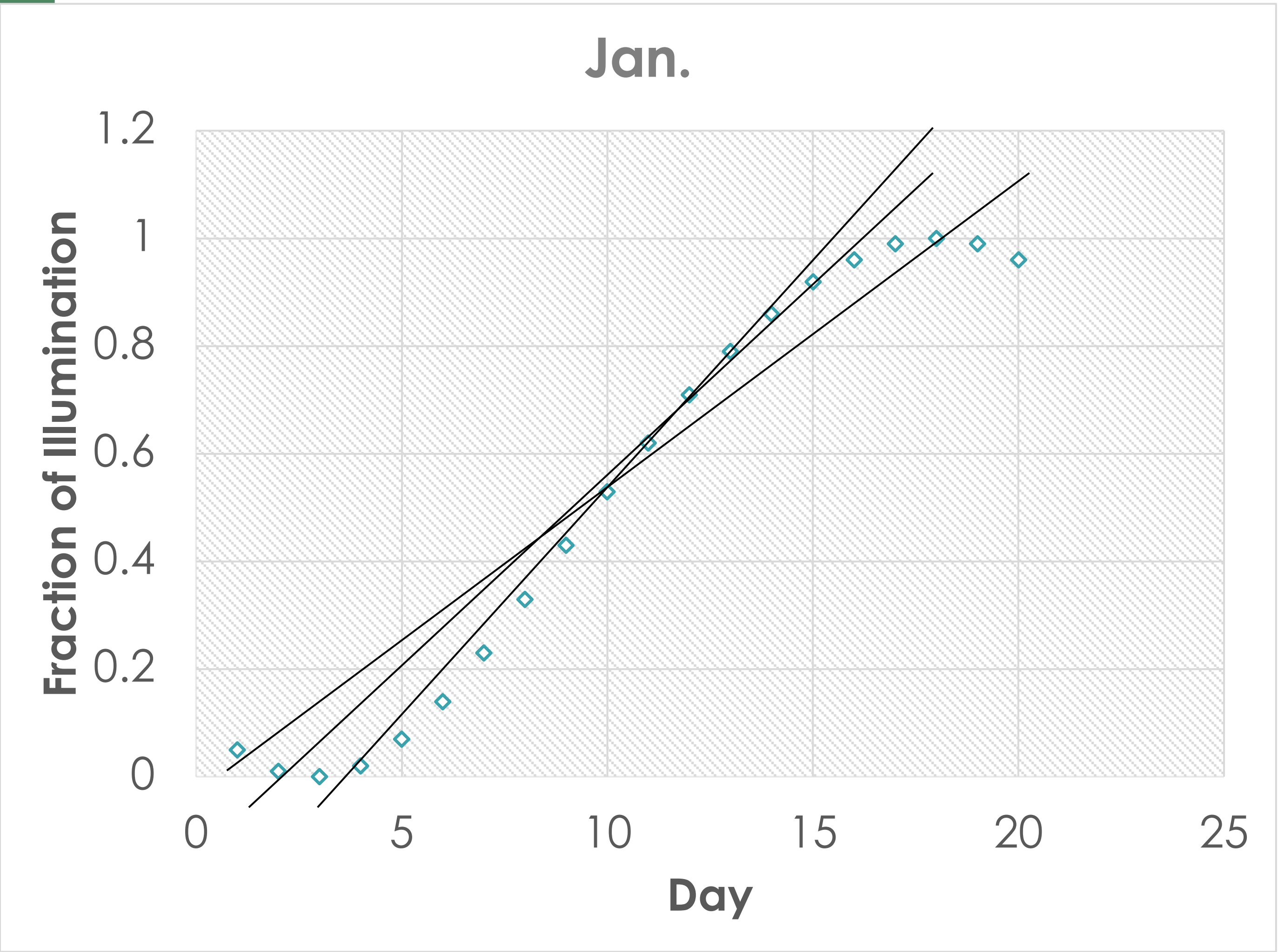
Predicting Moon Phases

Day	Jan.
1	0.05
2	0.01
3	0
4	0.02
5	0.07
6	0.14
7	0.23
8	0.33
9	0.43
10	0.53
11	0.62
12	0.71
13	0.79
14	0.86
15	0.92
16	0.96
17	0.99
18	1
19	0.99
20	0.96



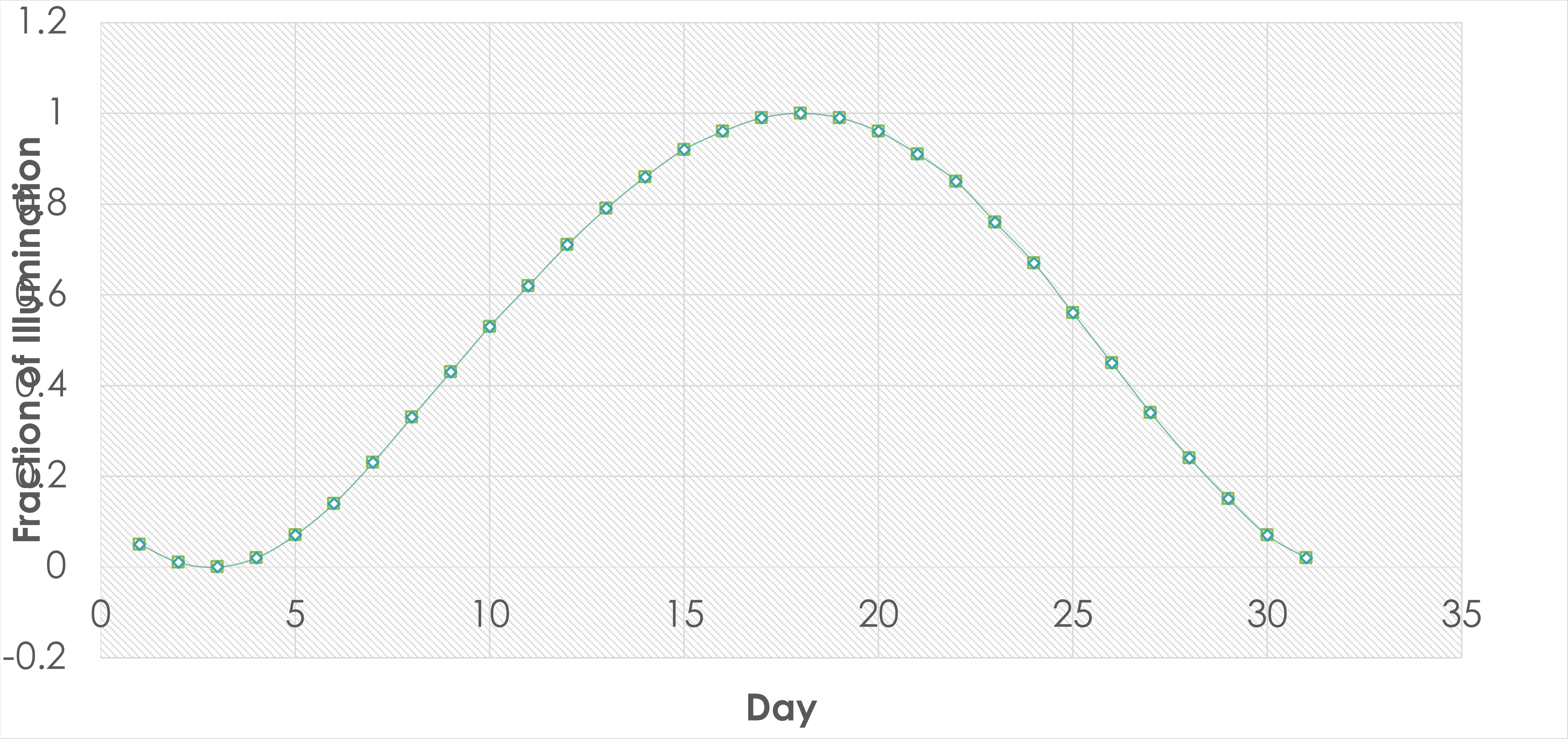
Oak Ridge National Laboratory

Linear Regression



- Looks like a linear fit would do a good job in predicting
- But what line? Say, y = ax + b.
- We have two parameters [a, b]. How to estimate them?
- The parameters that gives us the minimum error in predicting
- Minimize MSE = \sum (a(x_i) + b(y_i) + c - y)^2

Need for a more complex model

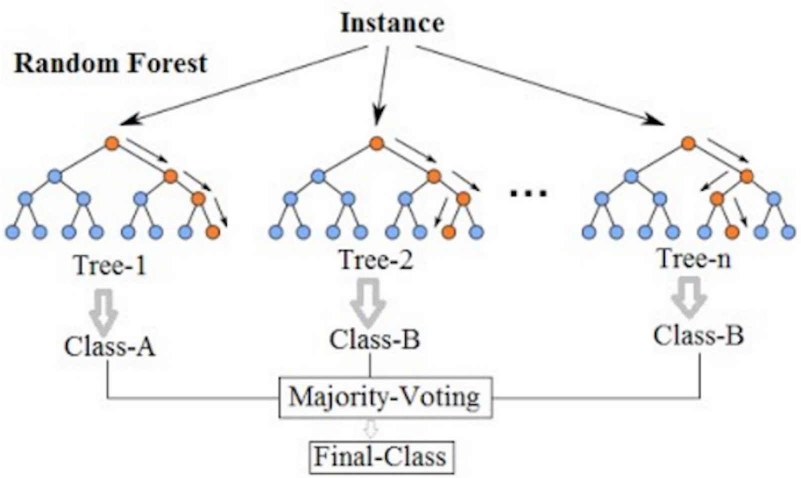


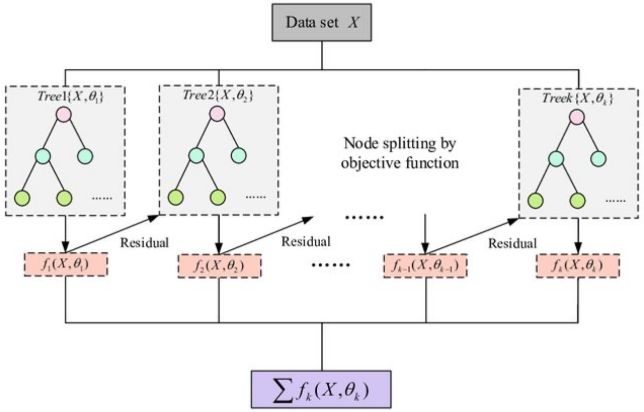
Why do we need complex ML Models?

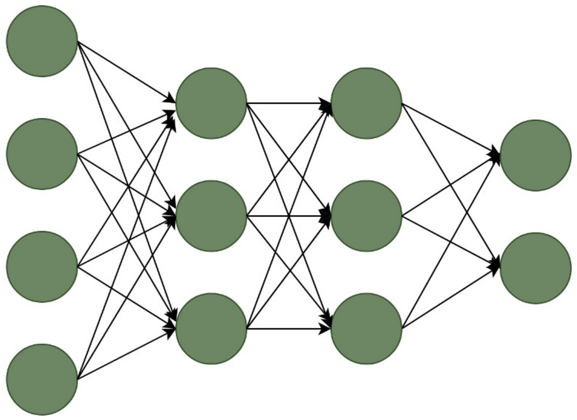
- We could fit a high-degree polynomial curve, but what's the degree?
- Not all data would show a familiar shape (like parabola in this case)
- Data points can have hundreds of features
- What if we need to predict the path of tornado or segment an image?

Complex Machine Learning Models

- Random Forest: combines outputs of multiple decision trees to reach a single result.
- XG-Boost: Iteratively build an ensemble of decision trees, each new model is trained to correct the mistakes made by the previous models.
- Artificial Neural Network



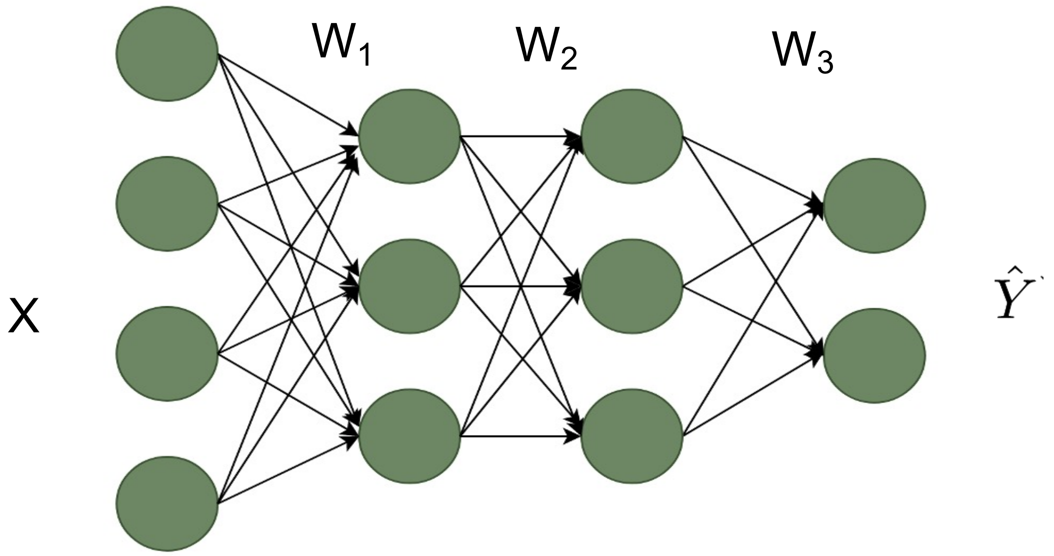




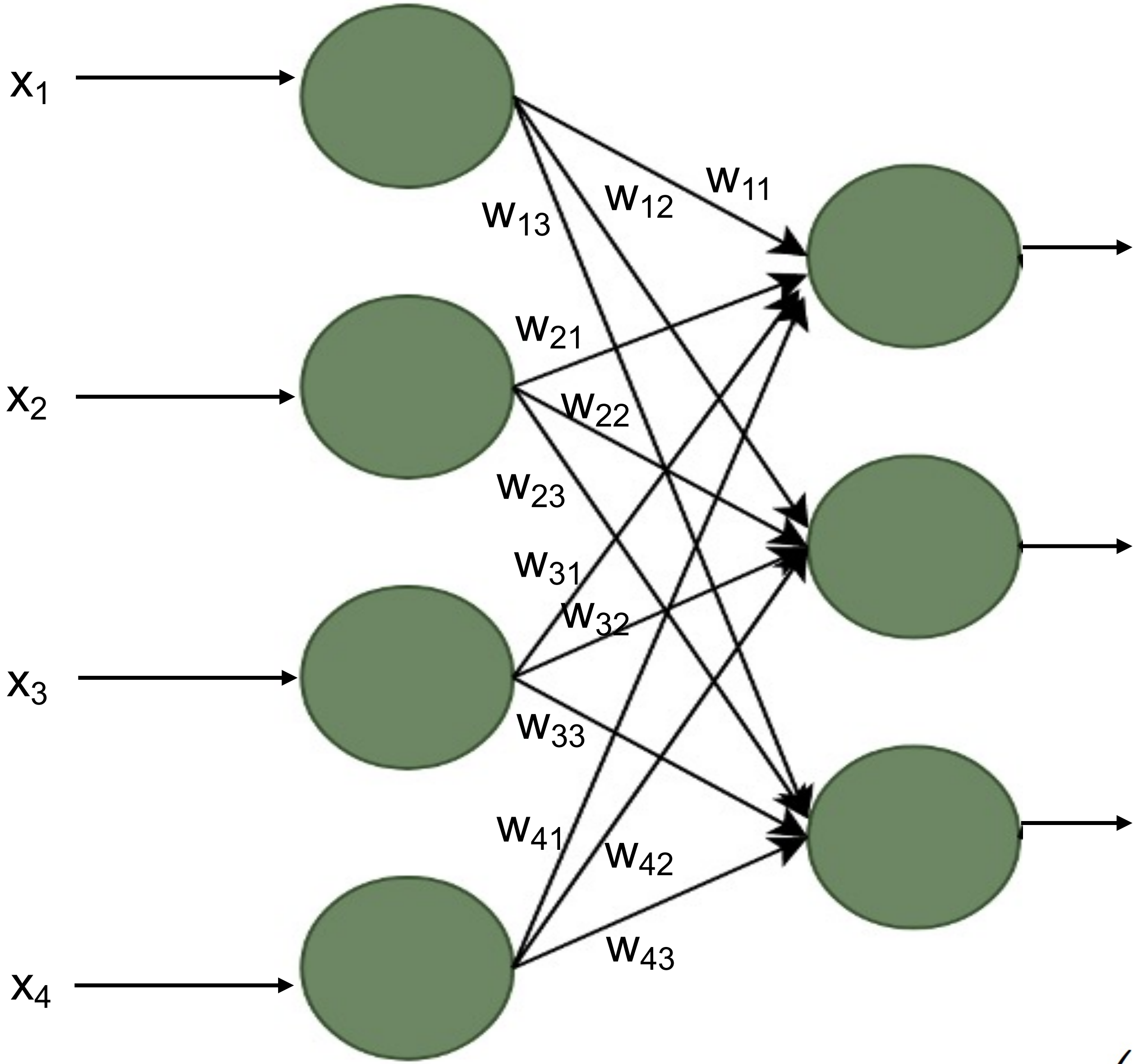
Deep Learning as Universal Function Approximator

- We have observations X, we want to predict Y. We want to learn f, where Y = f(X)
- For linear regression Y = f(X) = AX + B
- But, what if we don’t to make an assumption about the nature of the function (linear, polynomial, …)?
- Artificial Neural Network (ANN) can approximate and learn this f from available observation without any explicit assumption about f

Internals of an Artificial Neural Network



$$X = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}, W_1 = \begin{bmatrix} w^1_{11} & w^1_{12} & w^1_{13} \\ w^1_{21} & w^1_{22} & w^1_{23} \\ w^1_{31} & w^1_{32} & w^1_{33} \\ w^1_{41} & w^1_{42} & w^1_{43} \end{bmatrix}, W_2 = \begin{bmatrix} w^2_{11} & w^2_{12} & w^2_{13} \\ w^2_{21} & w^2_{22} & w^2_{23} \\ w^2_{31} & w^2_{32} & w^2_{33} \end{bmatrix}, W_3 = \begin{bmatrix} w^3_{11} & w^3_{12} \\ w^3_{21} & w^3_{22} \\ w^3_{31} & w^3_{32} \end{bmatrix}$$



$$\sigma(x_1 w_{11} + x_2 w_{21} + x_3 w_{31} + x_4 w_{41})$$

$$\sigma(x_1 w_{12} + x_2 w_{22} + x_3 w_{32} + x_4 w_{42})$$

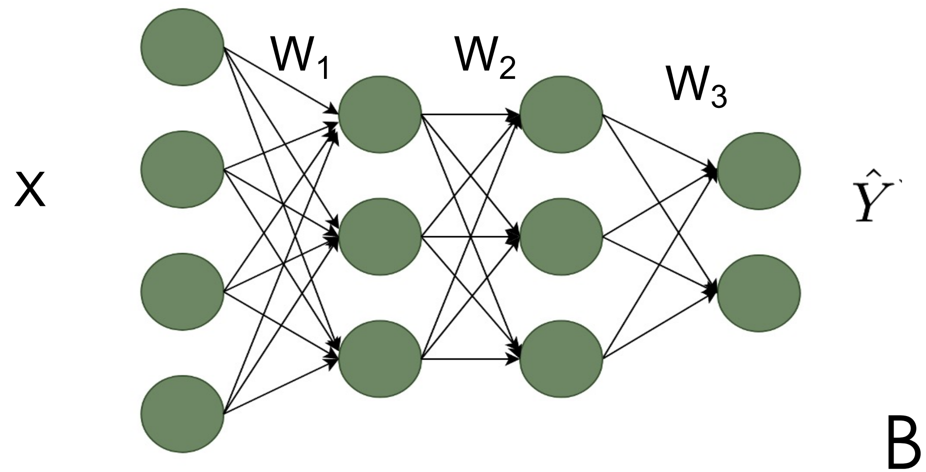
$$\sigma(x_1 w_{13} + x_2 w_{23} + x_3 w_{33} + x_4 w_{43})$$

$$O_1 = \sigma\left(\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}\right) \times \begin{bmatrix} w_{11}^1 & w_{12}^1 & w_{13}^1 \\ w_{21}^1 & w_{22}^1 & w_{23}^1 \\ w_{31}^1 & w_{32}^1 & w_{33}^1 \\ w_{41}^1 & w_{42}^1 & w_{43}^1 \end{bmatrix}$$

$$= \sigma(X \times W_1)$$

edit

Training a Neural Network



- Forward Pass

$$O_1 = \sigma(X \times W_1)$$

$$O_2 = \sigma(O_1 \times W_2)$$

$$\hat{Y} = O_3 = \sigma(O_2 \times W_3)$$

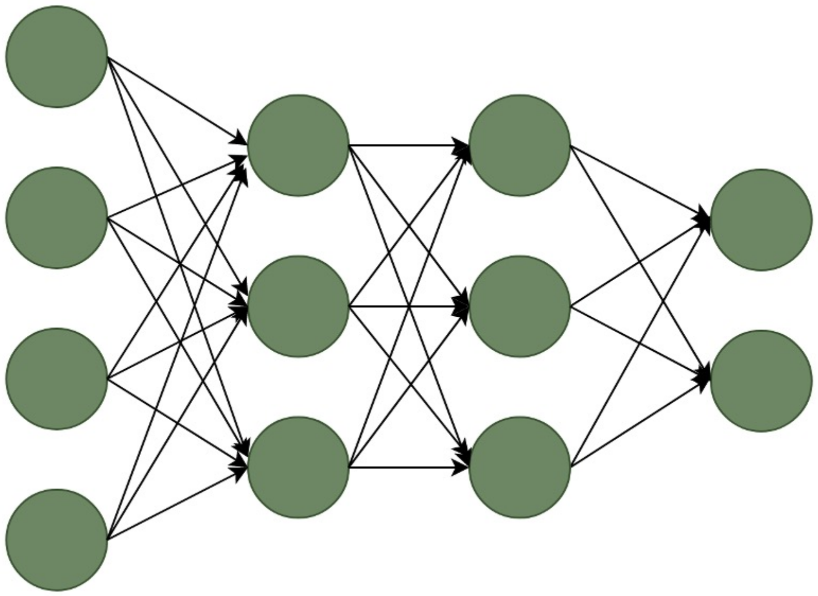
Backward Pass

$$L = Loss = CrossEntropyLoss(Y, \hat{Y})$$

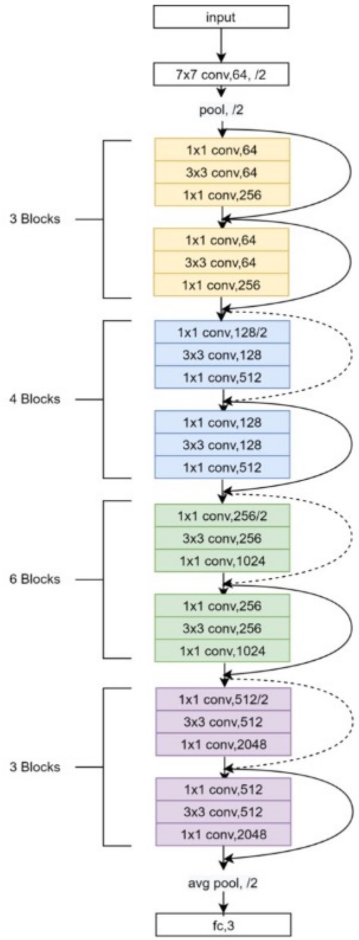
$$G = Gradient = \frac{\delta L}{\delta W}$$

$$W' = W - \eta \times \frac{\delta L}{\delta W}$$

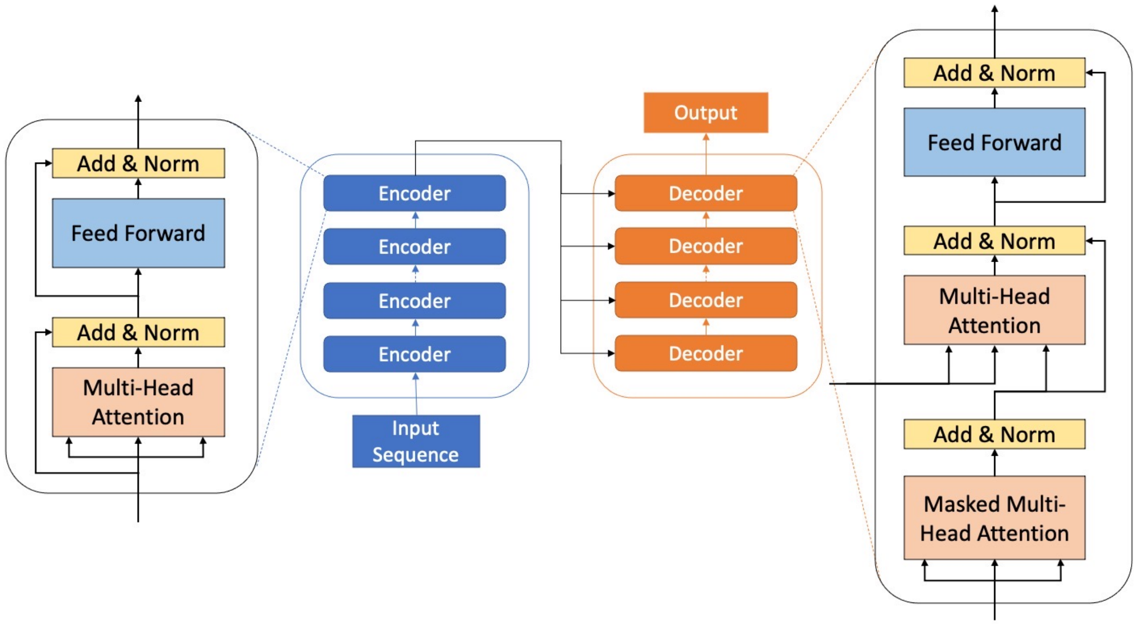
Deep Neural Networks (Deeper Networks)



ANN



ResNet50



Transformer

Steps to train a Deep Learning Model

- Load and Explore the Data
  - Create a Dataset object
  - Create a Dataloader object
- Design a Model
  - ANN/CNN/Modify an existing model
- Design Optimizer and Loss function
  - Optimizer decides how to update the model, loss gives the measure of error
- Train the model
  - For a predefined number of epochs/iterations
- Evaluate the model
  - Don’t update the parameters

```
train_dataset = NPZDataset(train_data_dir)
test_dataset = NPZDataset(test_data_dir)
train_dataloader = torch.utils.data.DataLoader(train_dataset, …)
test_dataloader = torch.utils.data.DataLoader(test_dataset, …)
```

```
class CNN(nn.Module):
    def __init__(self):
        …
    def forward(self, x):
        …
model = CNN()
```

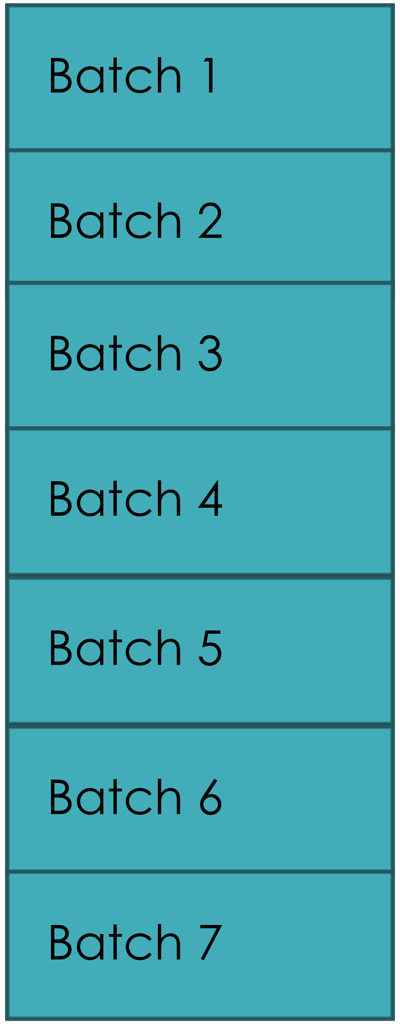
```
num_classes = 231
model = models.resnet50(pretrained=True)
num_ftrs = model.fc.in_features
model.fc = nn.Linear(num_ftrs,
num_classes)
```

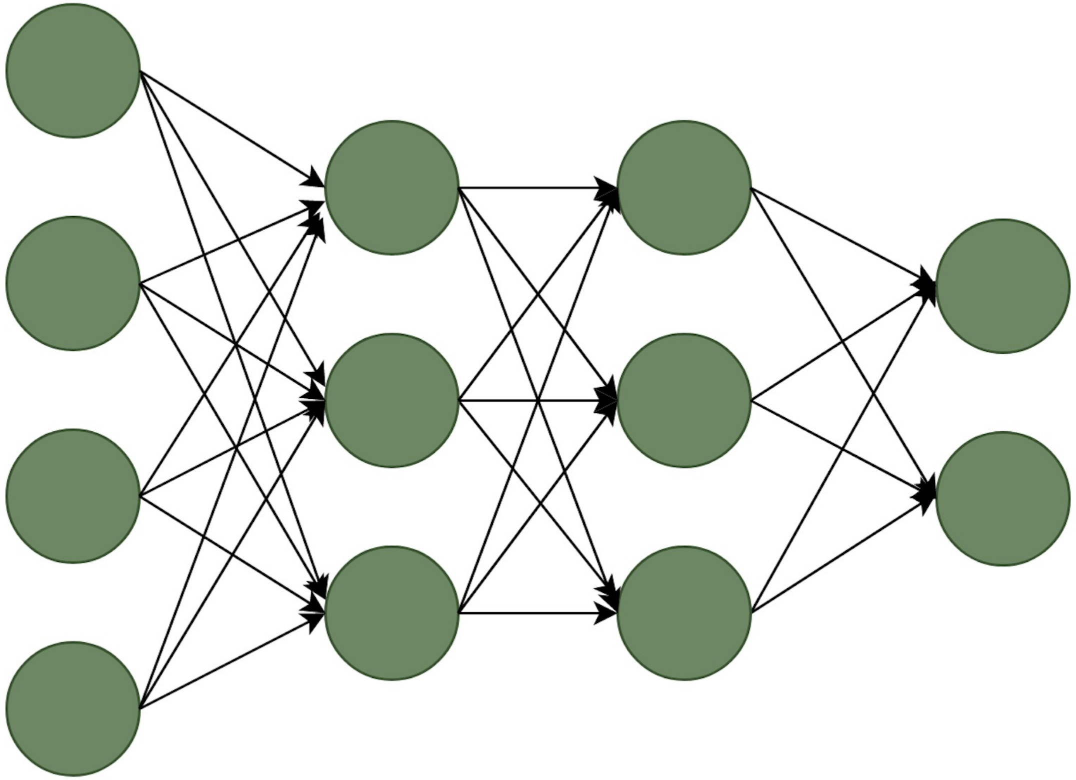
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

SCALING

```
for epoch in range(1):
    for i, data in enumerate(train_dataloader):
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        optimizer.step()
```

Training DL Model With Data Batches

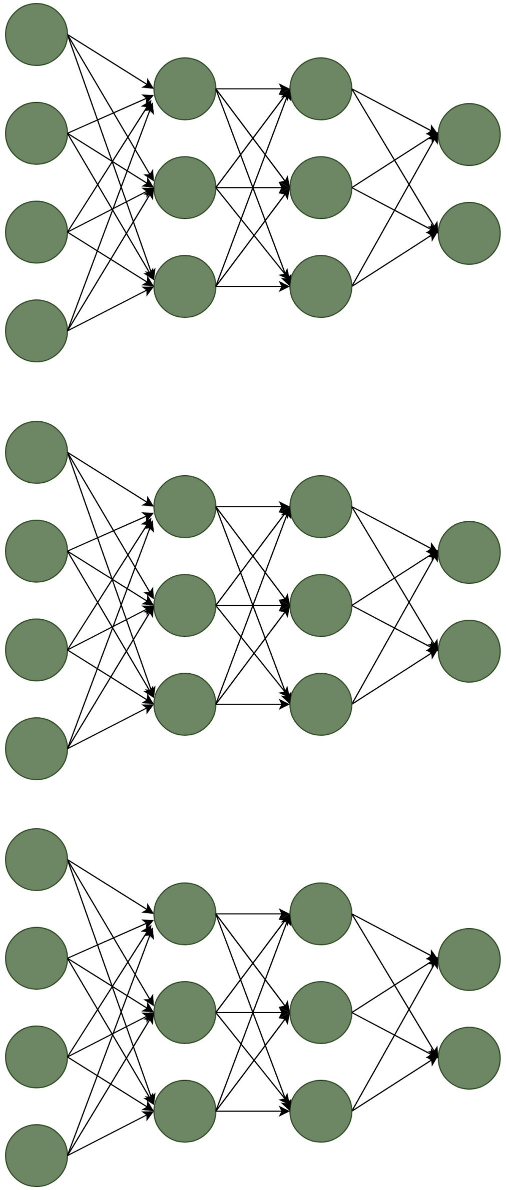




https://www.deeplearningbook.org/ Chapter 6, 8

Training DL Models with Large Data

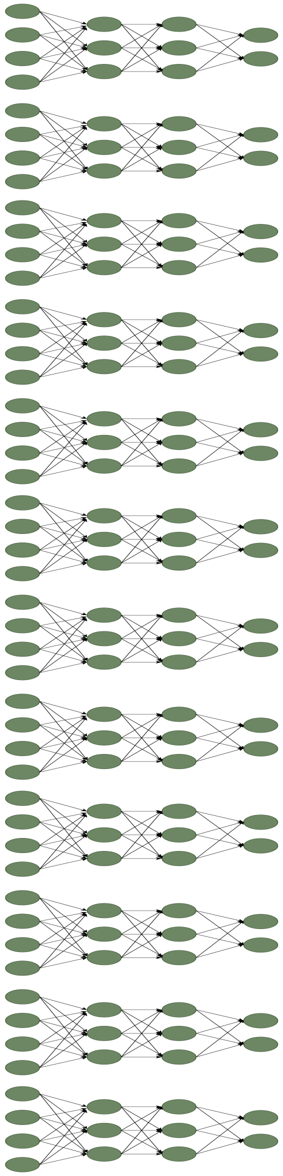
Batch 1.1
Batch 1.2
Batch 1.3



[Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour]

Scaling Up/Scaling Out

Batch 1



https://github.com/horovod/horovod

Scaling *Up* and *Out* a model

- Move the model to GPU

```
device = "cuda"
model.to(device)
```

- Run on multiple GPU

```
model = DataParallel(model, device_ids = [0, 1, 2])
```

- Run on multiple Nodes

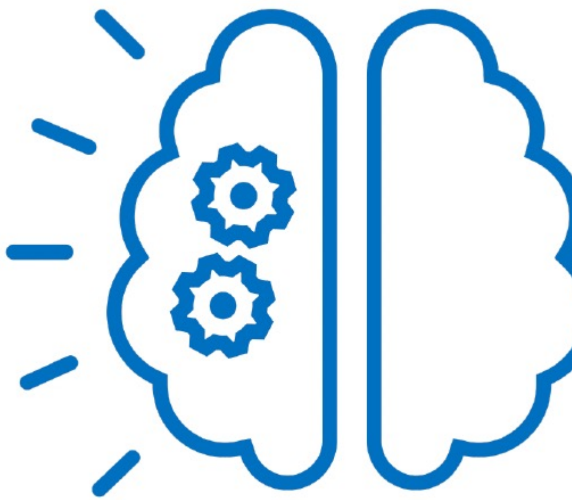
```
setup_DDP(backend="nccl")
model = DDP(model, device_ids = [0, 1, 2])
```

AI For Science Projects

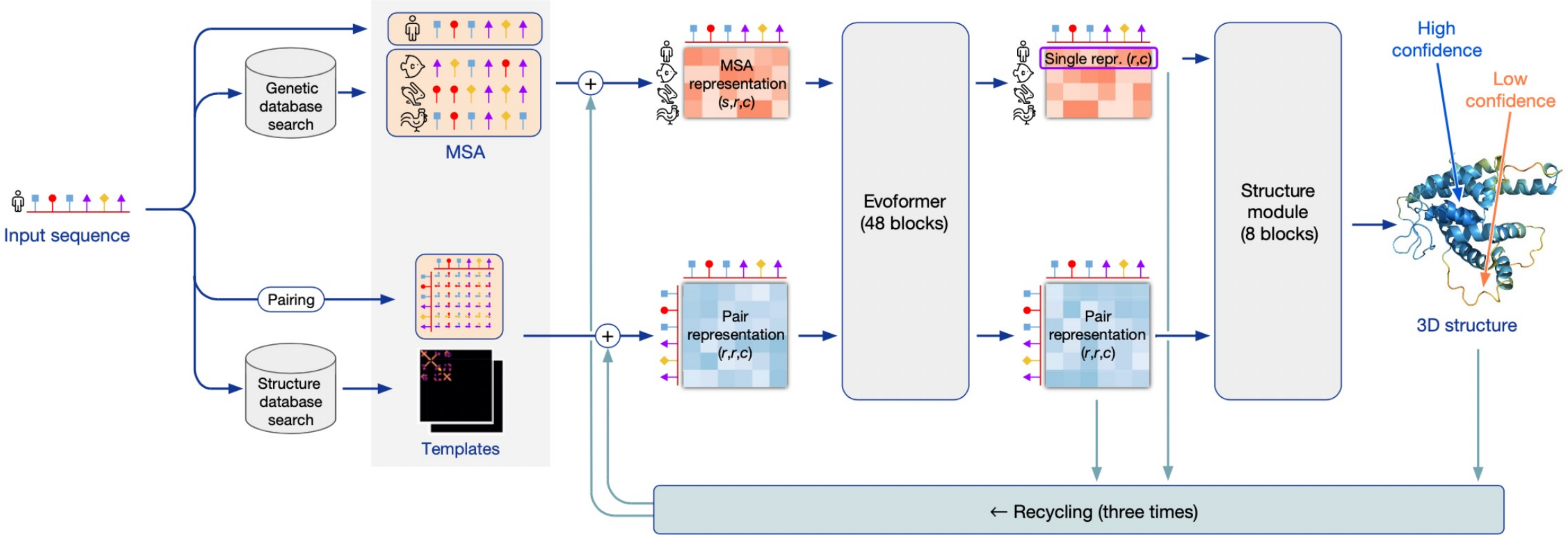
Project	AI Model(s)	Domain
AlphaFold for Protein Folding	Transformer w/ Cross Attention	Biochemistry
DeepThermo	Variational Auto Encoder	Material Science
CFD	Surrogate models (FCN, CNN, LSTM)	Fluid Dynamics
Language models for the prediction of SARS-CoV-2 inhibitors	Transformer (BERT)	Medicine/Biology

What is AlphaFold?

- A machine-learning-based model for predicting the 3D structure of proteins using only sequence as input
- Trained on known sequences and structures from the Protein Data Bank, as well as large databases of protein sequences

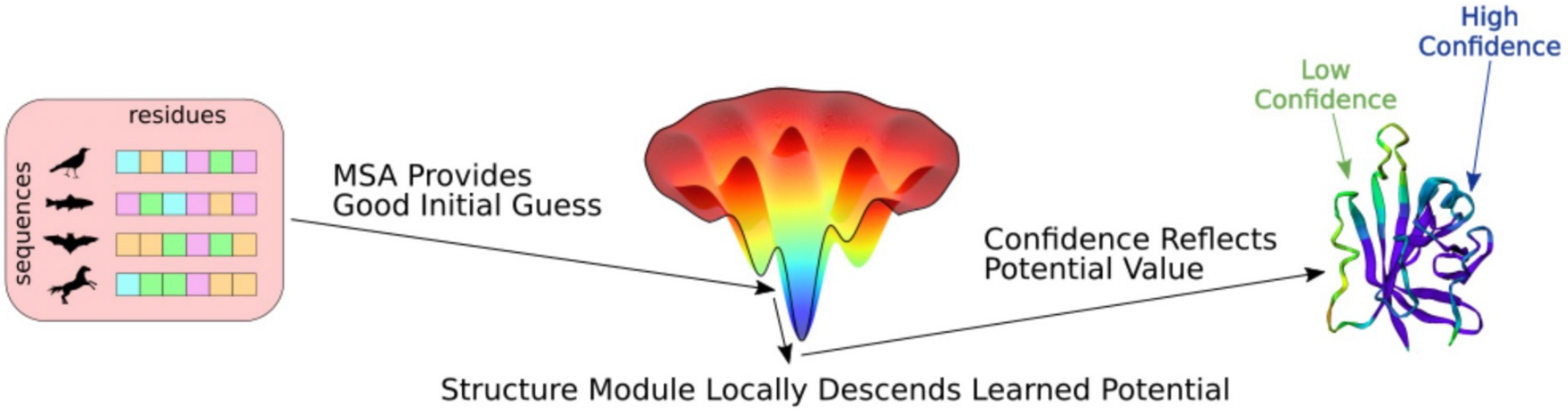


How does it work?



Jumper et al, Nature, 2021
See also Kendrew Lecture, 2021 (part 2): https://youtu.be/jTO6odQNp90

A guided search in a good energy function?

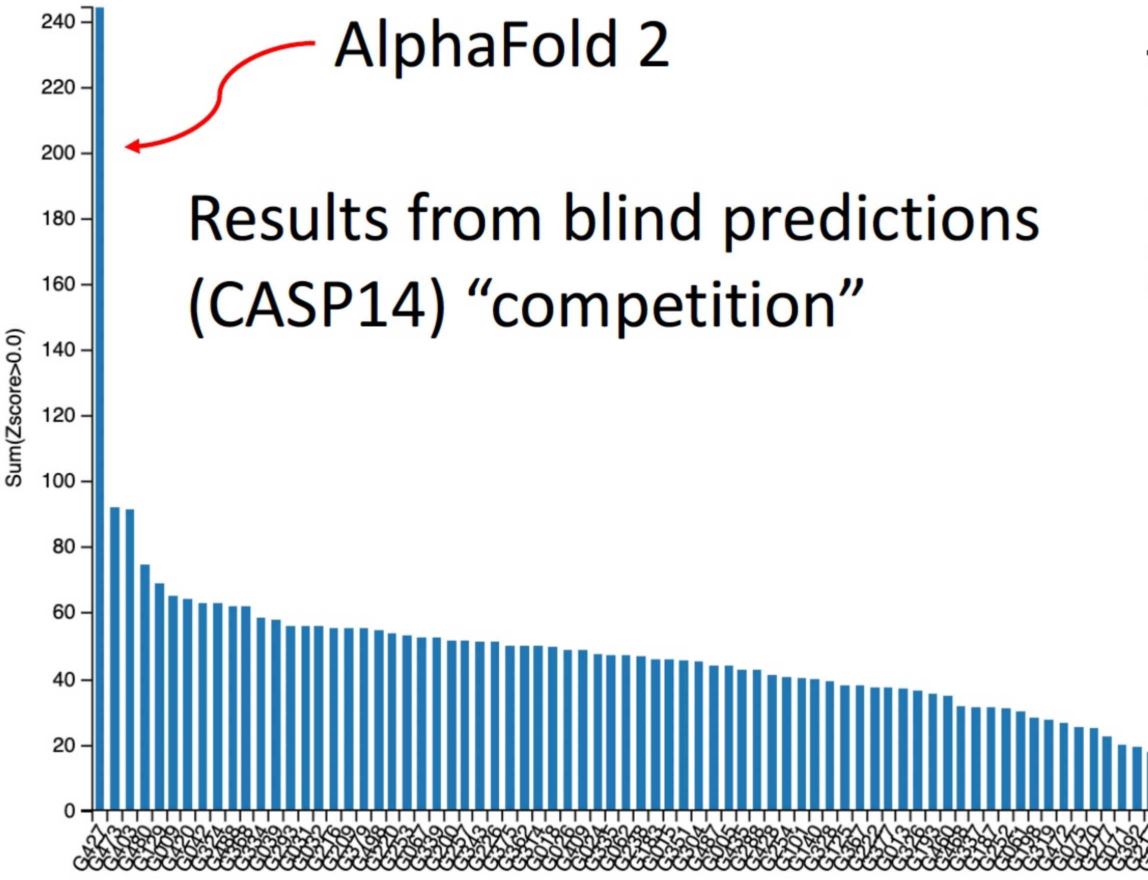
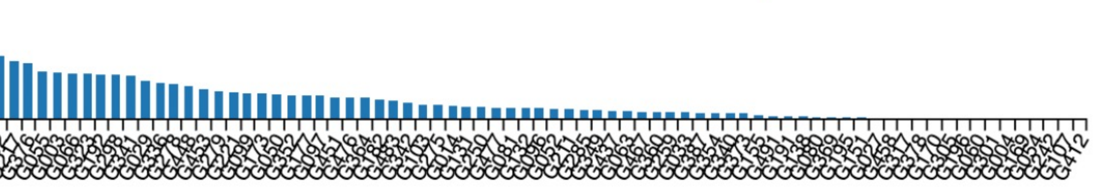


Roney and Ovchinnikov, bioRxiv, 2022

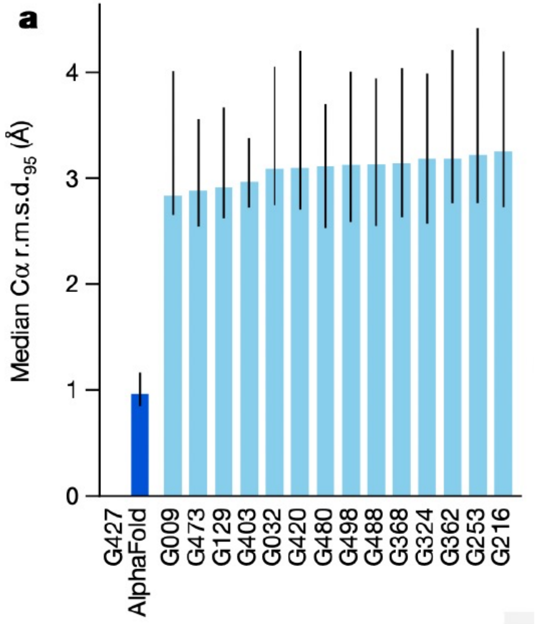
OK, but how well does it really do?

AlphaFold 2

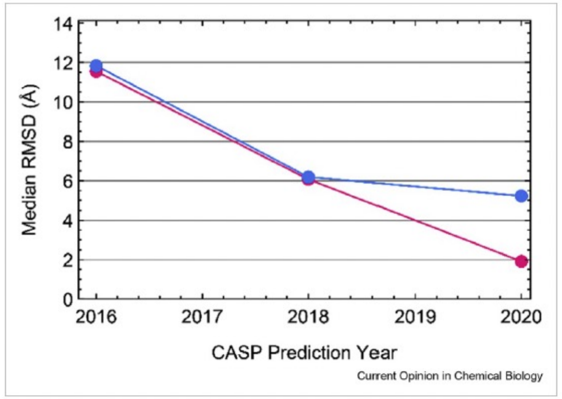
Results from blind predictions (CASP14) “competition”

Pereira et al, Proteins, 2021



Jumper et al, Nature, 2021



Accuracy of protein structure prediction. The median accuracy of the top two performing methods at CASP (Critical Assessment of protein Structure Prediction) is shown over the last four years. In CASP14 (late 2020), the AlphaFold2 system from DeepMind achieved ~2 Å accuracy over all heavy atoms for single domain apo proteins.

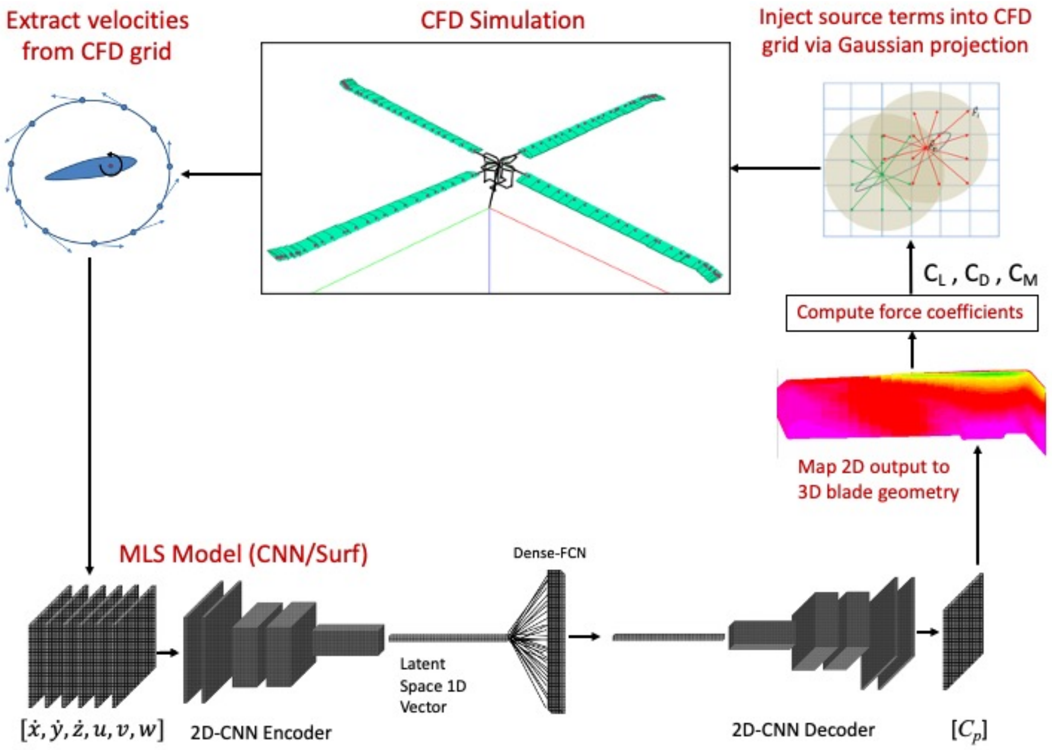
AlQuraishi,
Curr Opin Chem Biol, 2021

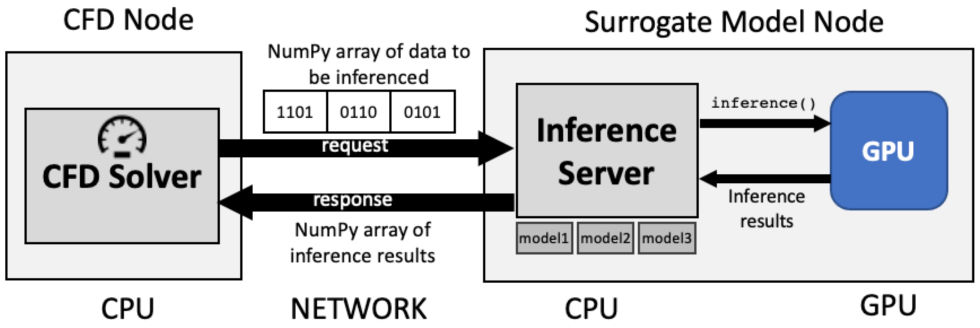
HPC-AI Execution Motifs

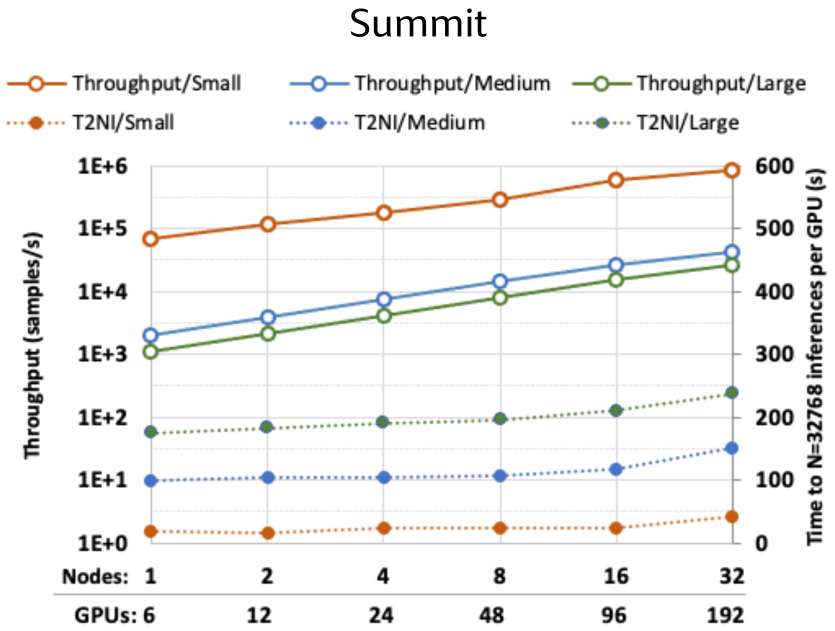
Execution Motif	Main Characteristics	Typical Bottlenecks	Example Use Cases
Steering	- Real-time interaction - Feedback loop between simulation and AI - Dynamic adjustments	- Data transfer between simulation and AI - Latency in decision-making	- Computational steering for adaptive mesh refinement - Real-time control of physical experiments
Multistage	- Sequential execution of tasks - Multiple AI and HPC components	- Data transfer between stages - Time spent in each stage	- Data pre-processing followed by training, and then analysis
Inverse Design	- AI-driven optimization - Iterative refinement of design parameters	- Convergence of optimization algorithm - Computationally expensive simulations	- Materials discovery - Drug design
Digital Twin	- Combining physics-based models and AI - Complementary strengths of both approaches	- Model integration and communication - Training physics-informed AI models	- Combining molecular dynamics with machine learning potentials - Weather prediction
Distributed Models	- Parallel execution of AI and/or HPC components - Scalability and efficiency	- Data and model parallelism overhead - Synchronization and communication	- Distributed training of large language models - Parallel execution of large-scale simulations
Adaptive Execution	- Dynamic resource allocation - Adjusting execution based on changing conditions	- Load balancing - Decision-making for resource allocation	- Adaptive mesh refinement in simulations - Auto-tuning of AI models

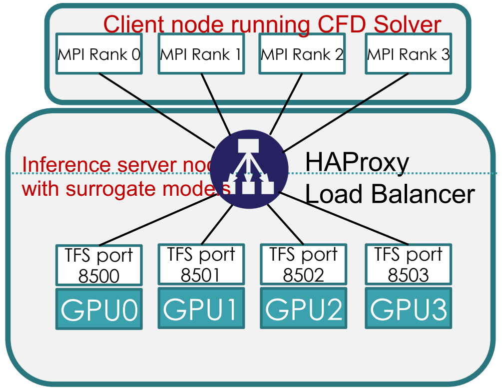
Gainaru and Jha et al. (2023)

Machine-Learned Rotorcraft Aerodynamics







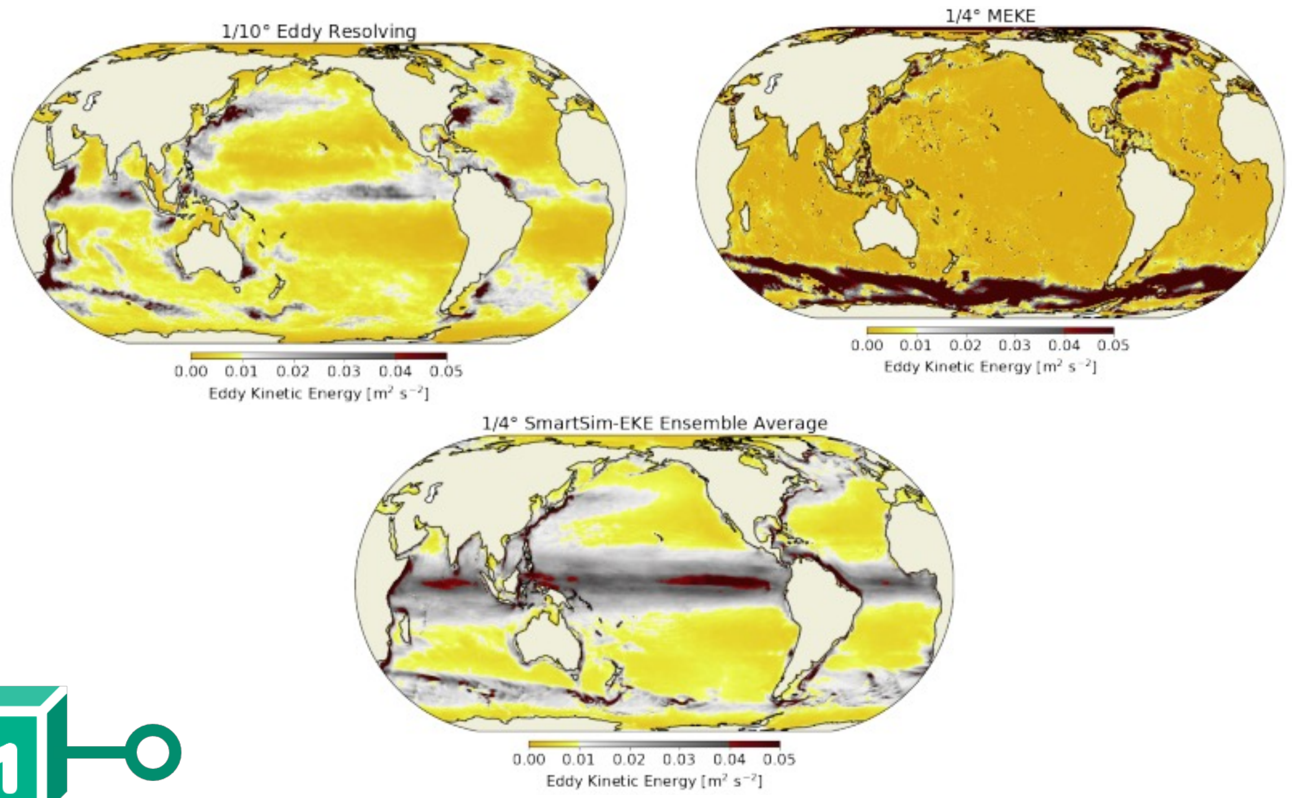


TensorFlow serving
↔ gRPC

Brewer et al., Production Deployment of Machine-Learned Rotorcraft Surrogate Models on HPC (2021)

Machine-Learned Turbulence Models

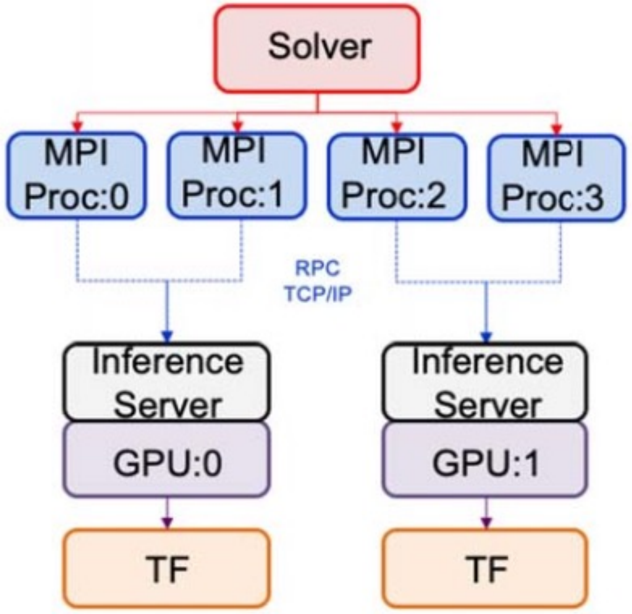
- In climate simulations, machine learning can be used as a surrogate model for eddy kinetic energy, radiation, or precipitation.
- This allows simulation on coarser grids with better resolution.
- Like the other CFD case, this requires many inferences.

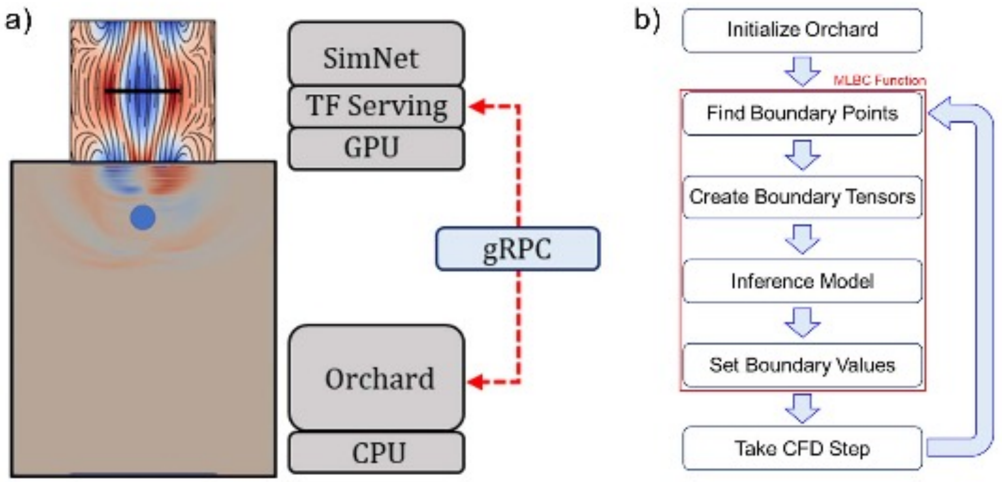


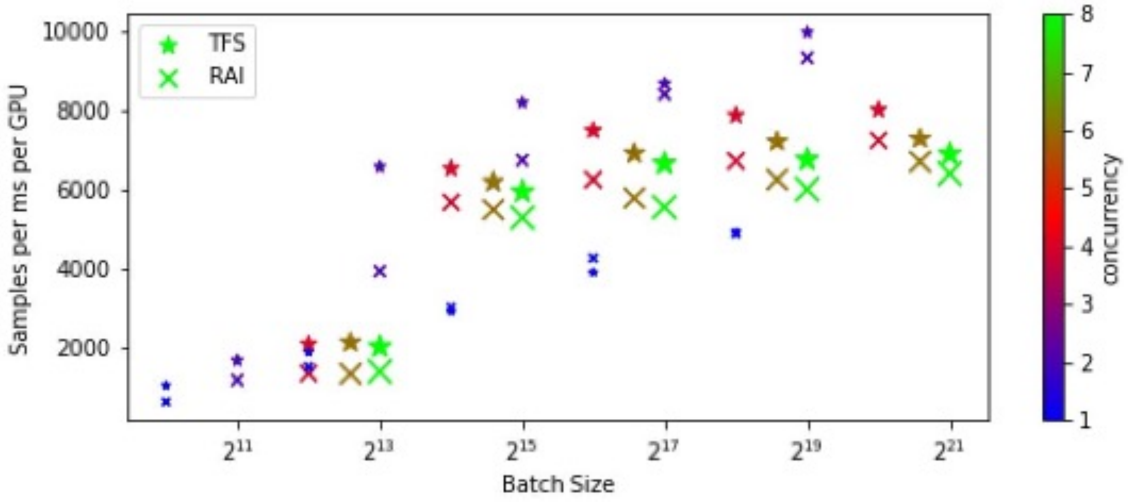
SMARTSIM

Partee, Sam, et al. "Using machine learning at scale in numerical simulations with SmartSim: An application to ocean climate modeling." *Journal of Computational Science* 62 (2022): 101707.

Machine-Learned Boundary Conditions



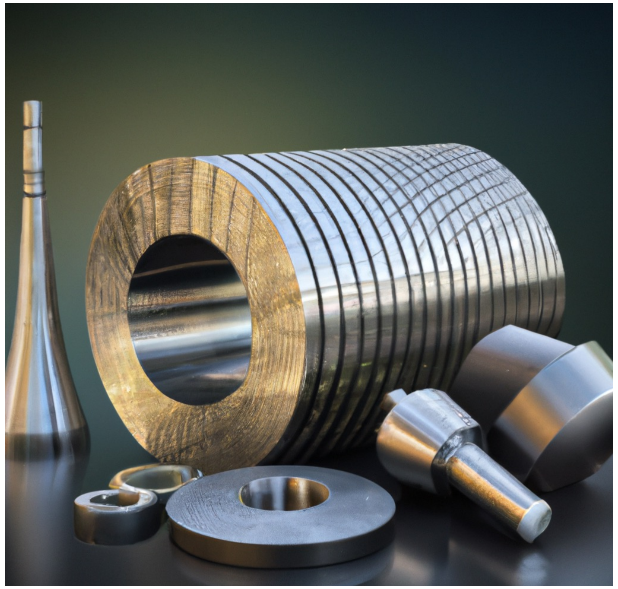




Boyer et al., Scalable Integration of Computational Physics Simulations with Machine Learning (2022)

Metropolis Monte Carlo

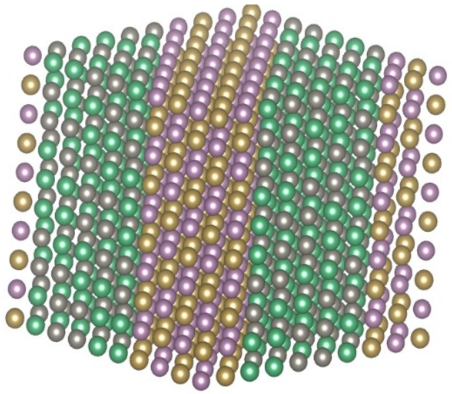
- Goal: sample configurations of a complex system given a specified distribution.
  - E.g. average E of an alloy at temperature T?



Designable Materials

Airplane engine

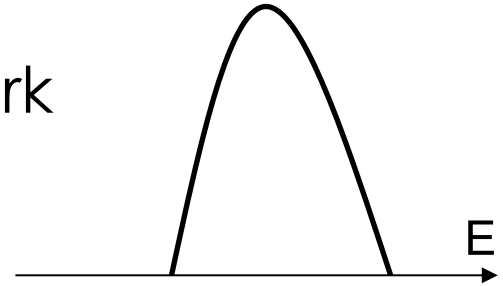
Mo Nb Ta W



Configuration: **X**

$$< E > = \sum_x E(X) * P(X)$$

- Random Sampling won't work
- But, if generated $\{X_i\}$ follows,



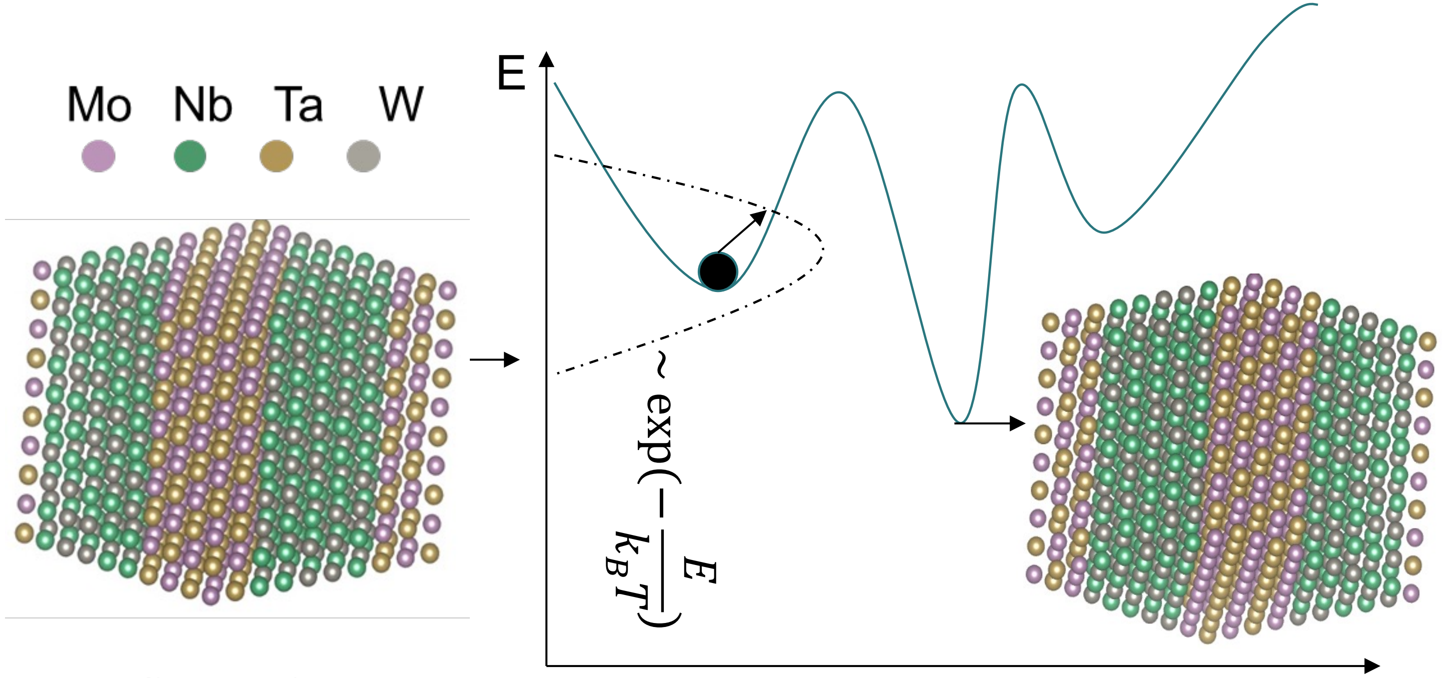
$$P \sim \exp(-\frac{E}{k_B T})$$

- Then it becomes a simple average

$$< E > = \frac{1}{N}\sum_{i=0}^{N} E(X_i)$$

Metropolis Monte Carlo and Current SOTA – PT, WL

- Challenge **1**: local minima at low T -> partially resolved (PT & WL)

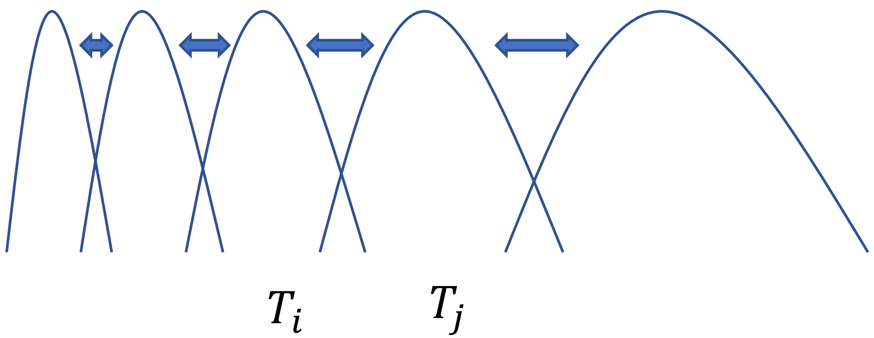


Configuration: **X**

MC proposal (X -> X`)

Challenge **2**

Parallel Tempering



$$A = \min[1, \exp(-\Delta)]$$

$$\Delta = (E_i - E_j)\left(\frac{1}{k_B T_j} - \frac{1}{k_B T_i}\right)$$

Challenge 2: -> still open

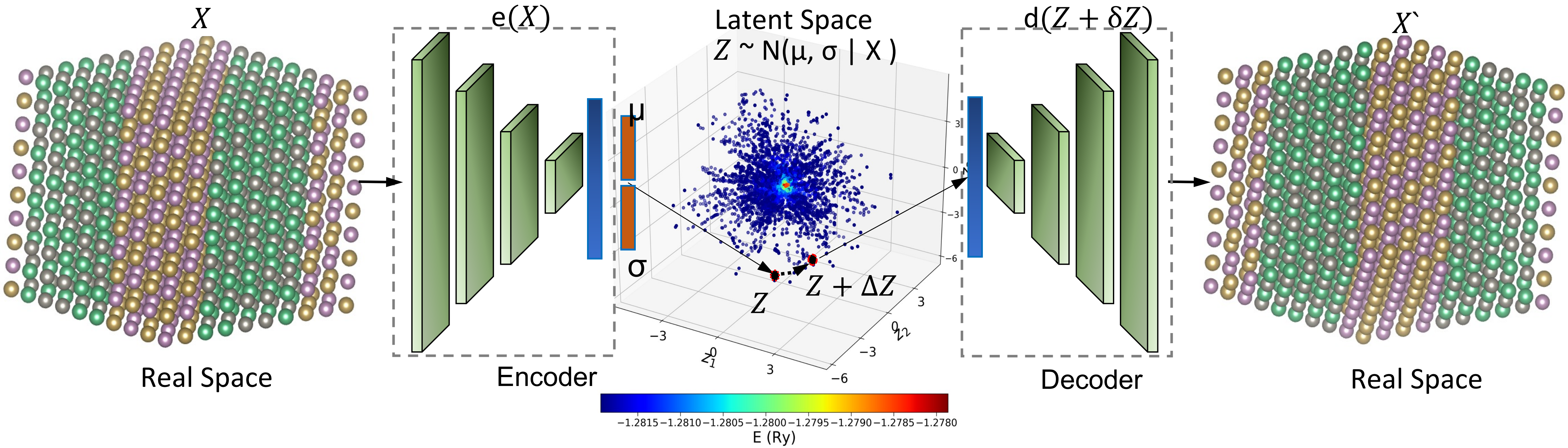
Scalable and generic MC proposal for faster convergence (time-to-solution)

$$E_b \xrightarrow{\text{MC proposal}} E_a$$

No recipe for global update

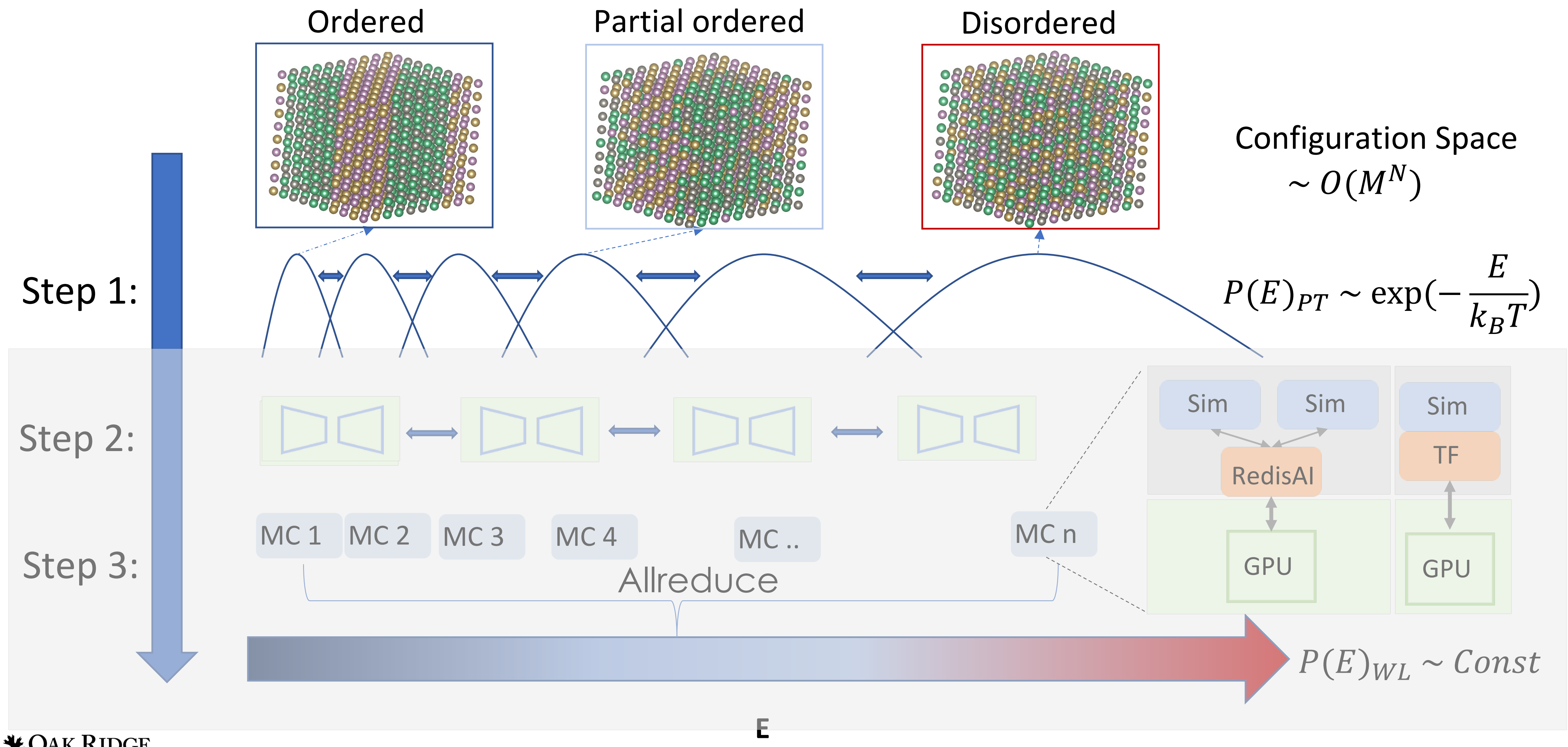
Deep Learning Generated MC proposal

- Address challenge 2: scalable and generic MC proposal

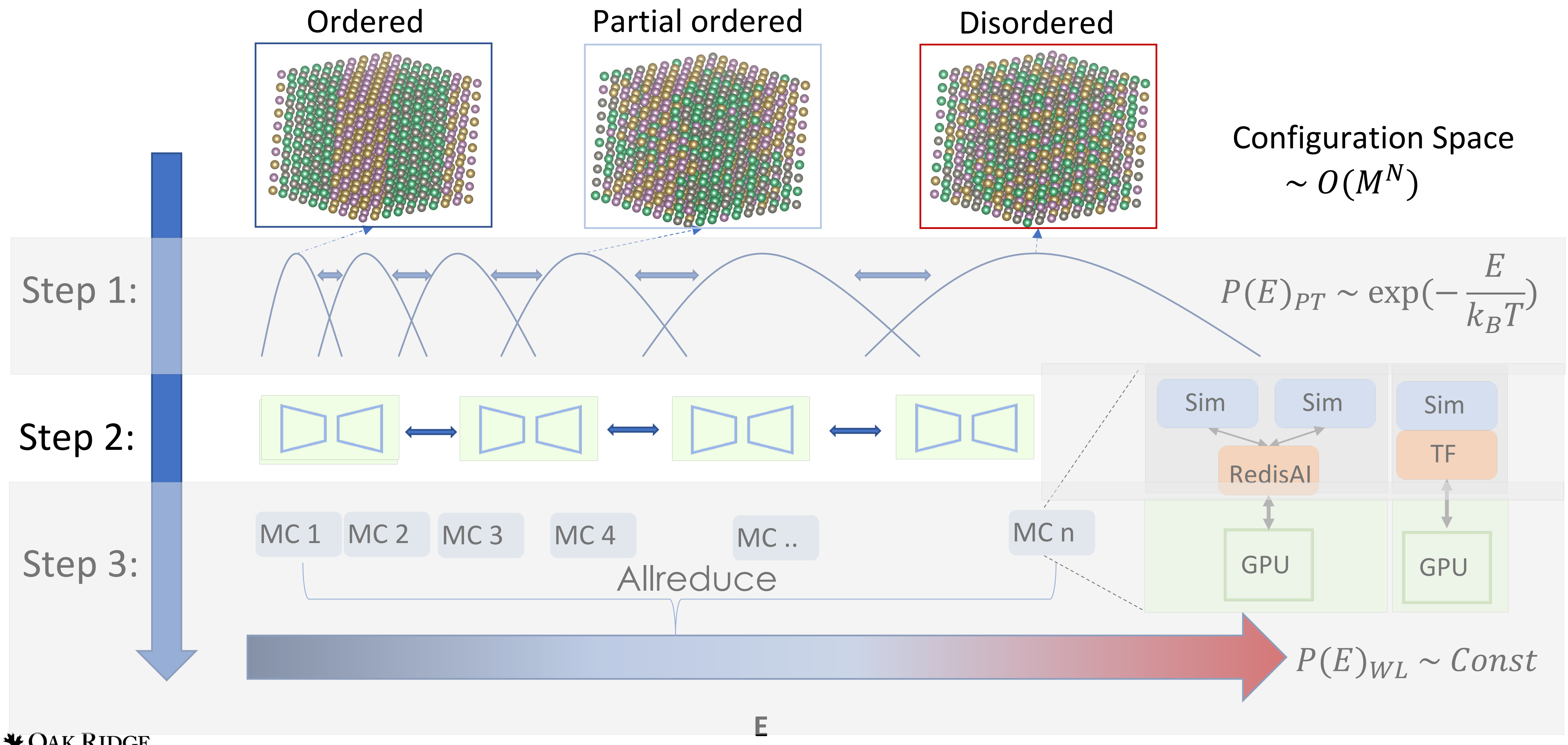


J. Yin, et al, DeepThermo: Deep Learning Accelerated Parallel Monte Carlo, IPDPS'23

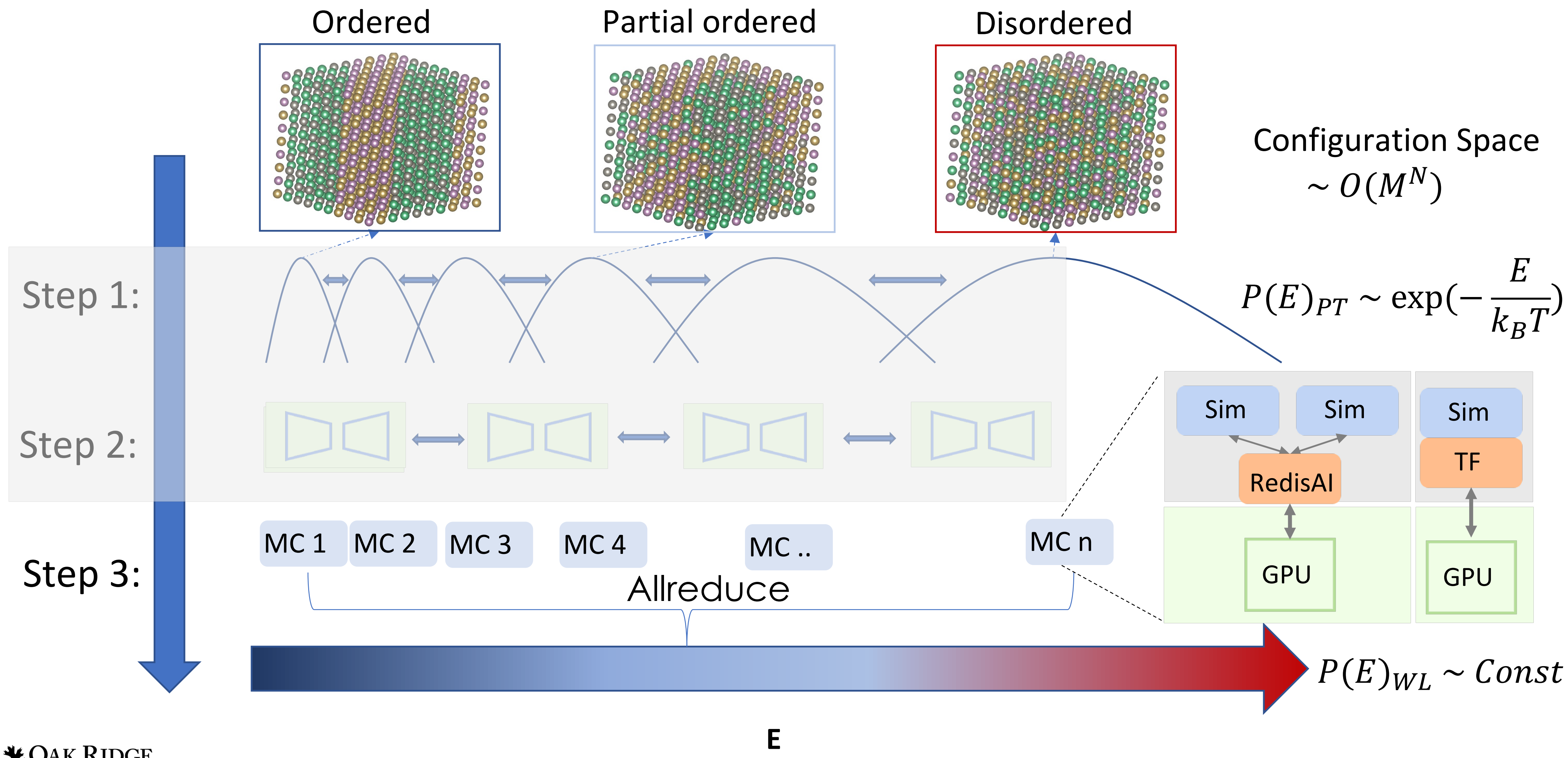
DeepThermo Approach



DeepThermo Approach

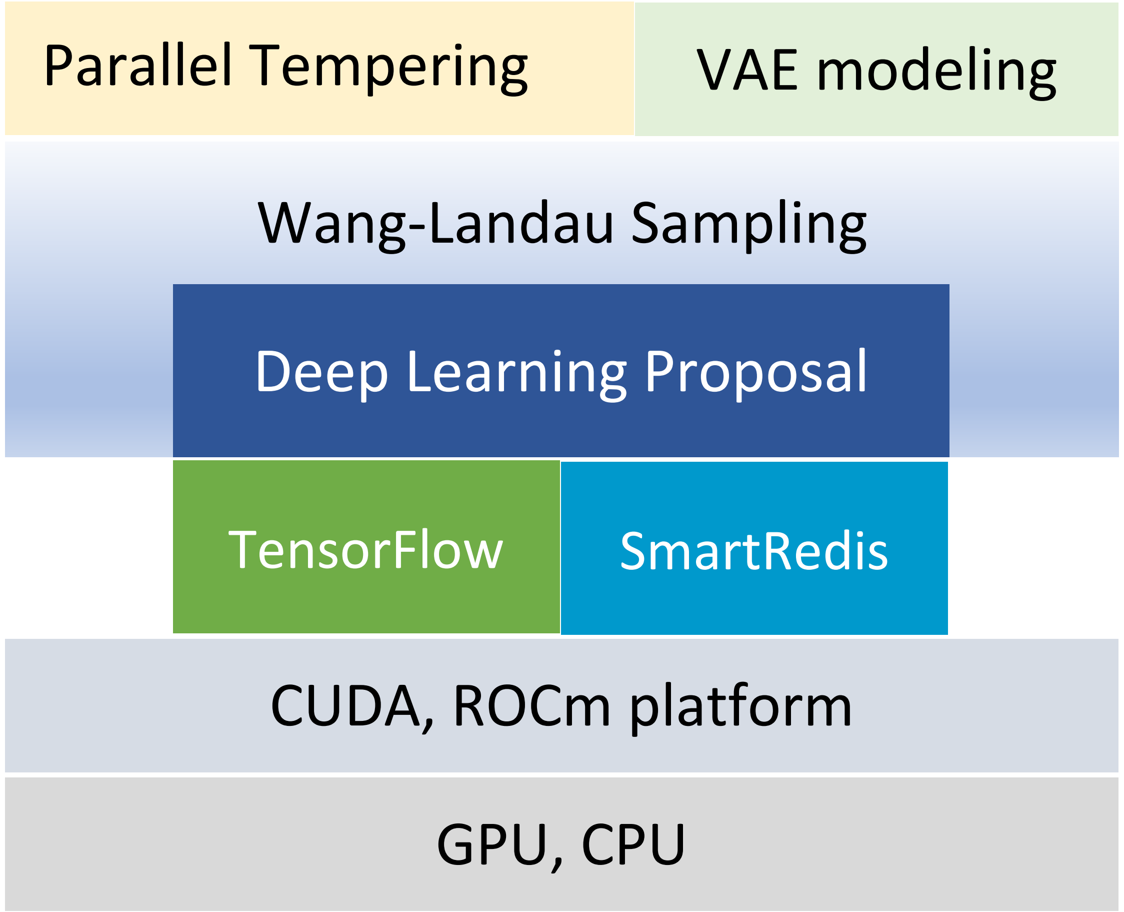


DeepThermo Approach



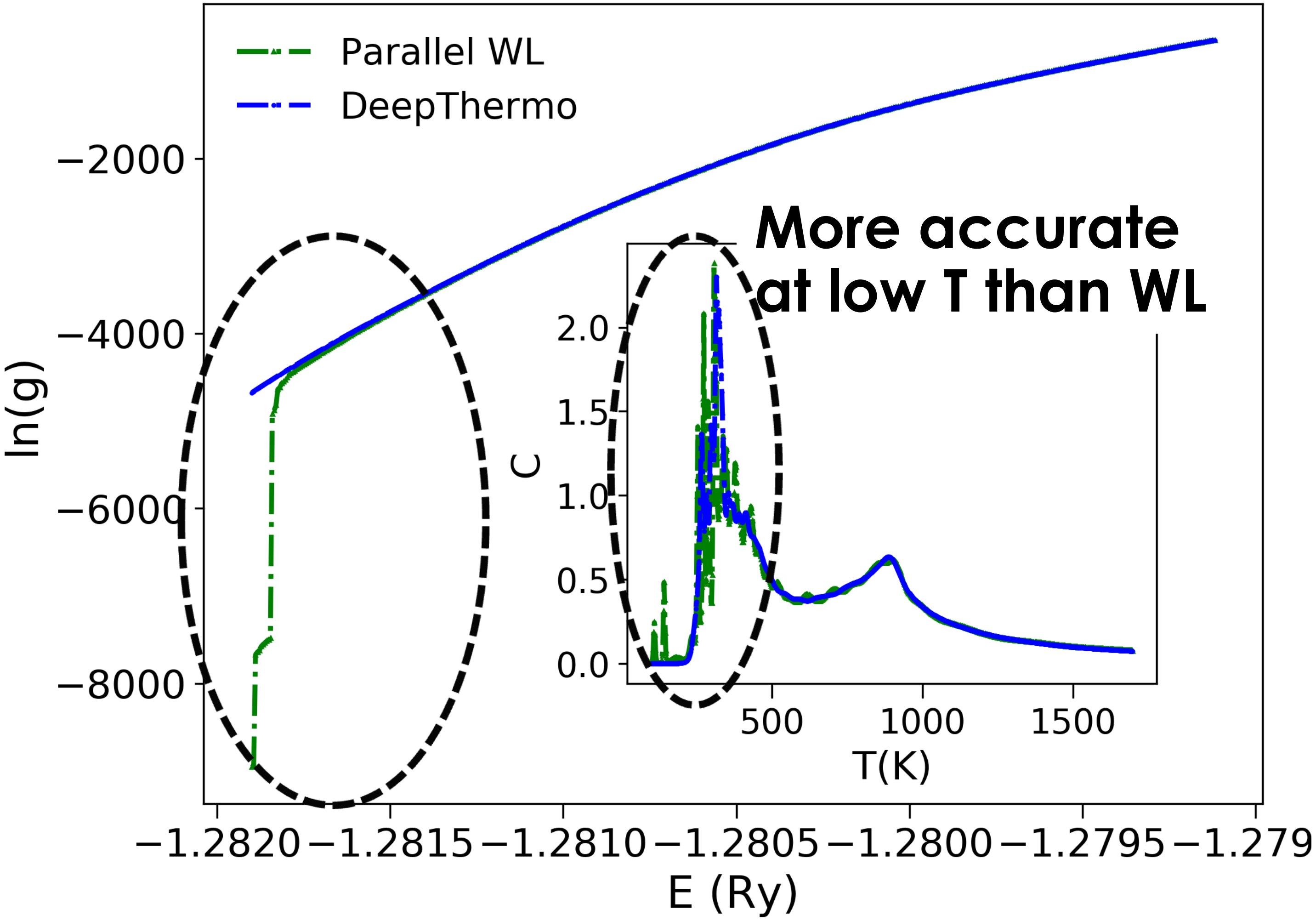
DeepThermo Architecture

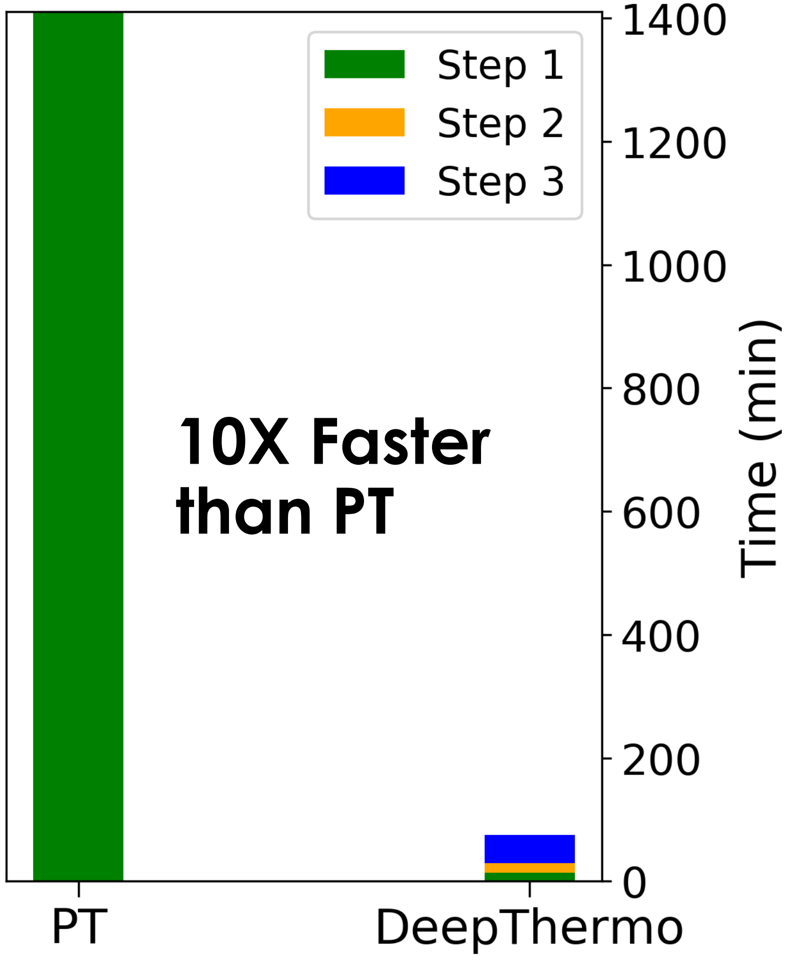
- Architecture
  - Application: PT, WL, VAE
  - Framework: TF, SmartRedis
  - DL stack: CUDA/ROCm
- Application layer is independent of DL stack
- Portability
  - Nvidia/AMD GPU, CPU



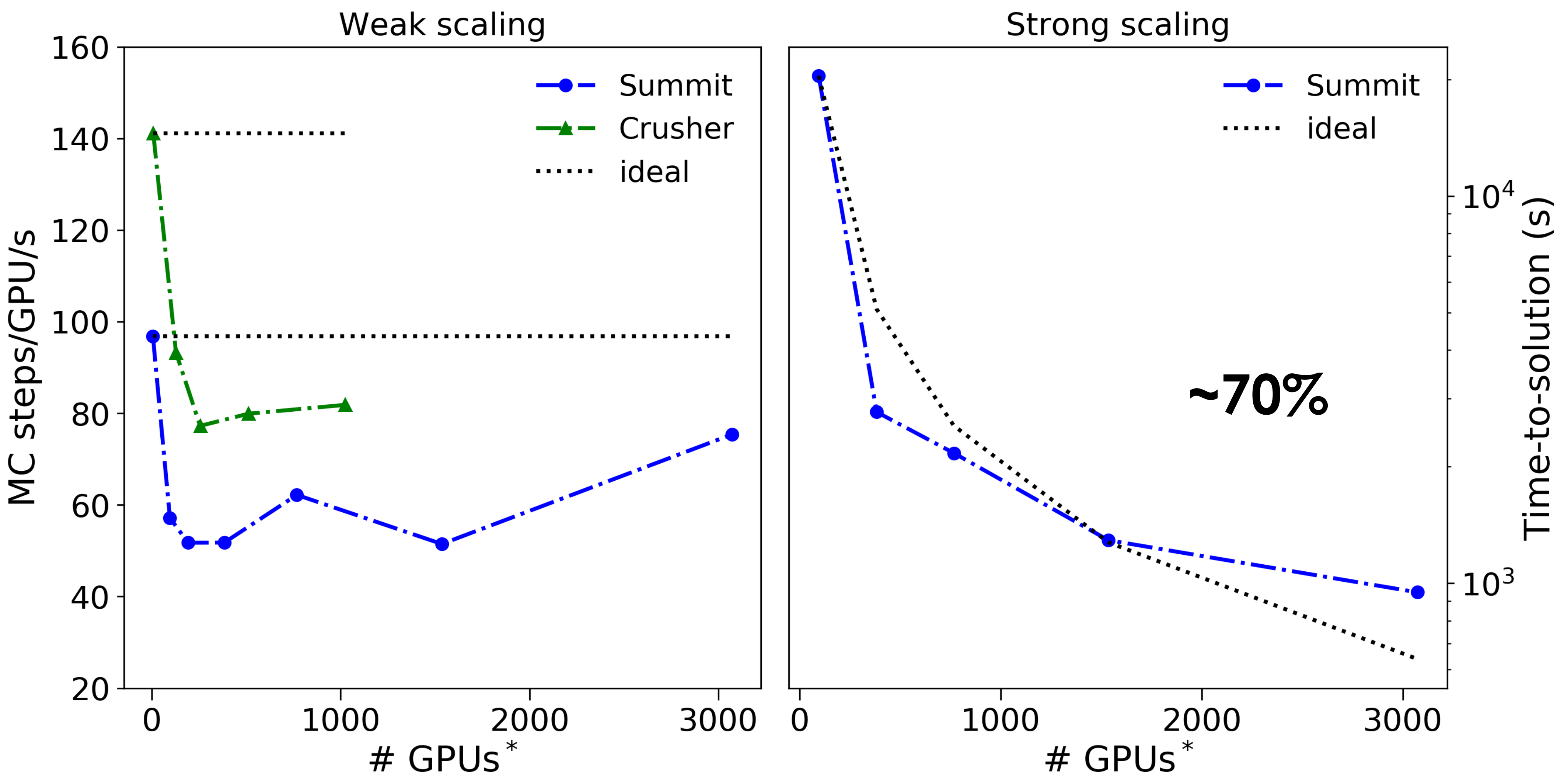
Comparisons with PT & WL

- Improvement on current SOTA





Scalability on Summit and Frontier (Crusher)



Throughput

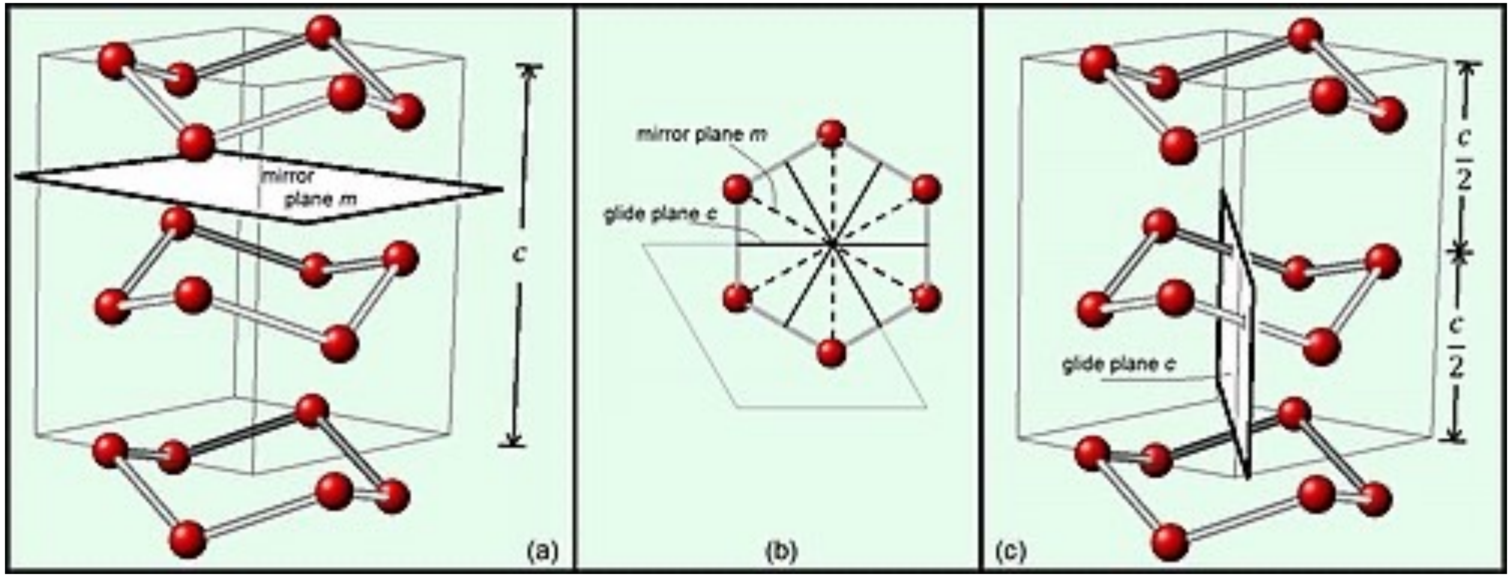
Time-to-solution

Outline (2nd Hour)

- Introduce spacegroup problem
- Visualize spacegroup data
- Build a simple ANN to perform classification
- Use ResNet50 to perform classification on a single GPU
- Make the code multi-GPU and run on single node (PyTorch's DataParallel)
- Make the code multi-GPU, multi-node

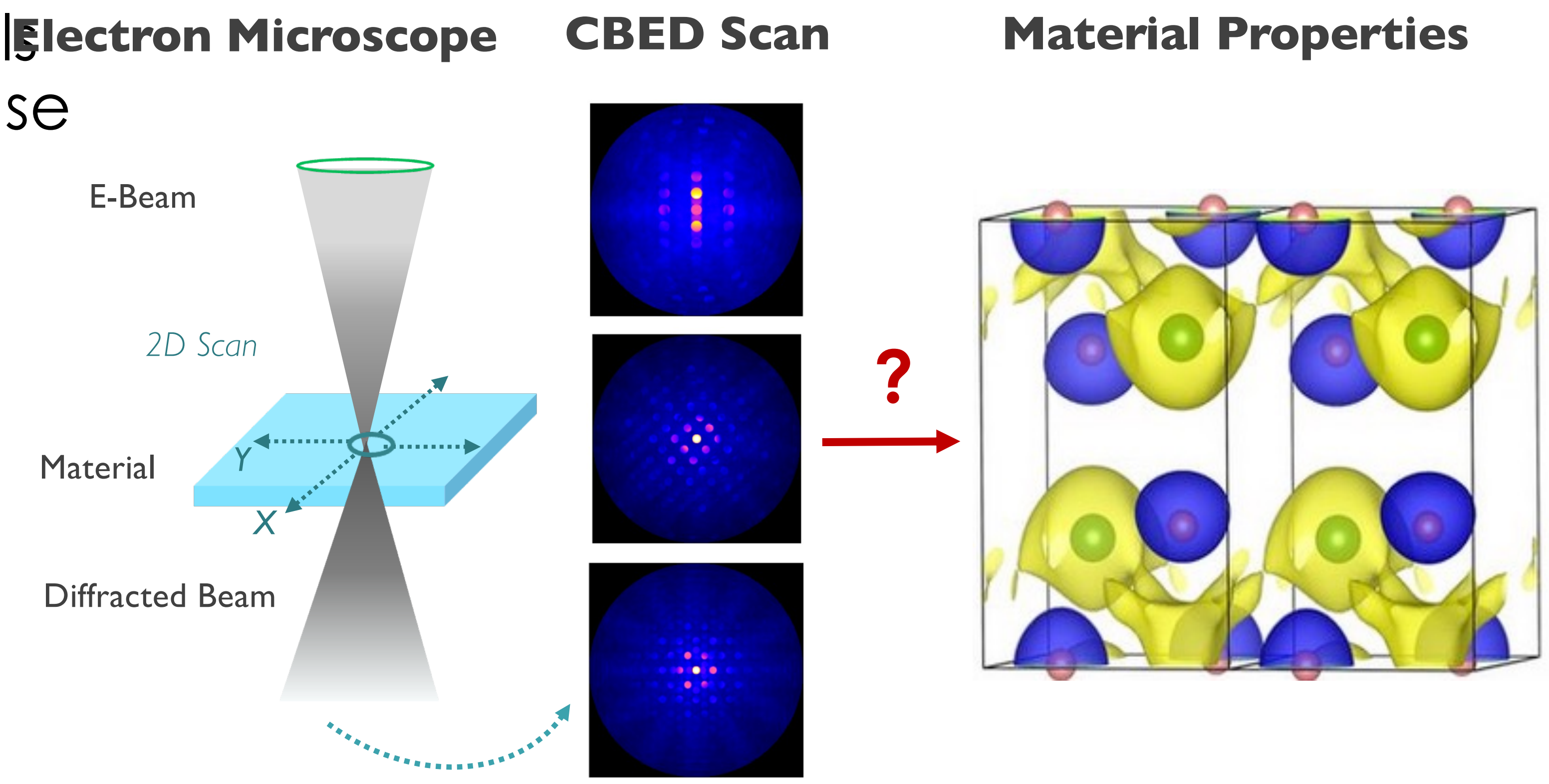
Case-study: Spacegroup Classification

- In mathematics, physics and chemistry, a space group is the symmetry group of an object in space, usually in three dimensions.



A Database of Electron Diffraction Patterns for ML of the Structural Properties of Materials

- Cover over 60,000 materials in material project database
- Labelled with crystallographic space groups, lattice constants/angles, etc
- Download link: (550GB) [10.13139/OLCF/1510313]
- [MLCommons Benchmarks]



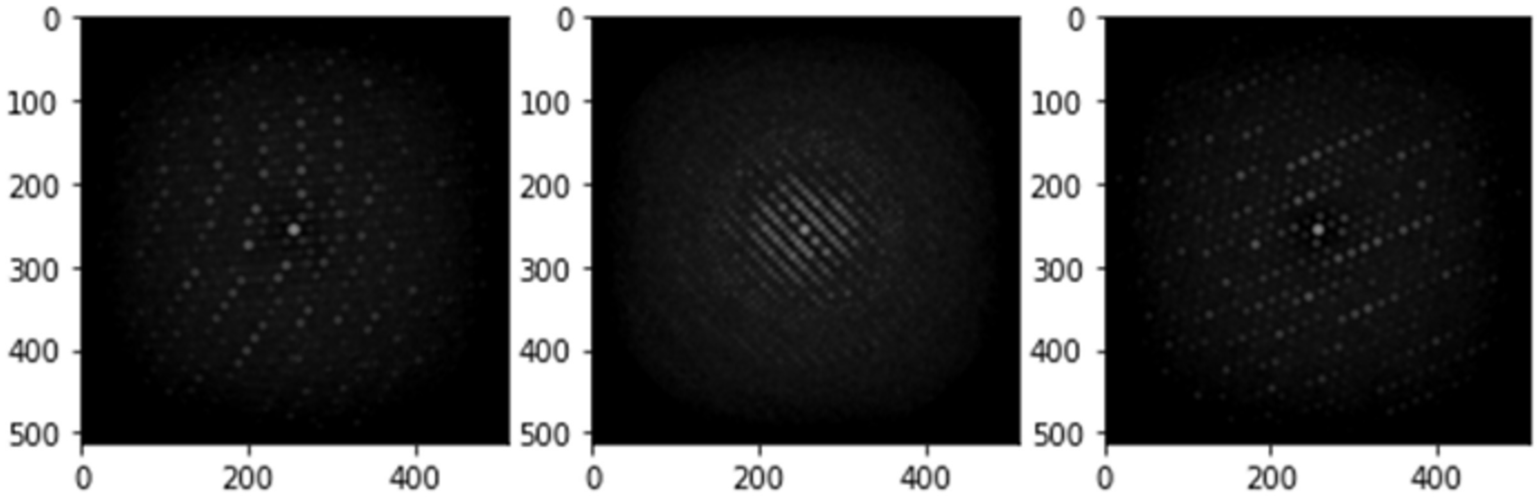
NAMSA: 10.11578/dc.20201001.90

A Database of Electron Diffraction Patterns for ML of the Structural Properties of Materials

- Metadata:

	filename	groupname	energy_keV	formula	material	space_group	attrs
0	batch_dev_112.h5	sample_1_0	100	K12 Fe4 O16	mp-505092	62	{'abc_angstrom': [7.764597170000000048, 9.2551...
1	batch_dev_112.h5	sample_1_1	110	K12 Fe4 O16	mp-505092	62	{'abc_angstrom': [7.764597170000000048, 9.2551...

- Image data:



PyTorch Dataset to Load the Data

- Data is saved as npz format
- Each file has two fields
- Three diffraction images were taken at slightly different angle for one sample

```
import numpy as np
from pathlib import Path
import torch
from torch.utils import data
import glob

class NPZDataset(data.Dataset):
    def __init__(self, npz_root):
        self.files = glob.glob(npz_root + "/*.npz")
        print("Number of files: ", len(self.files))

    def __getitem__(self, index):
        sample = np.load(self.files[index])
        x = torch.from_numpy(sample["data"])
        y = sample["label"][0]
        return (x, y)

    def __len__(self):
        return len(self.files)
```

Creating Dataset and DataLoader

```
train_data_dir = "/gpfs/alpine/world-shared/stf218/sajal/stemdl-data/train"
test_data_dir = "/gpfs/alpine/world-shared/stf218/sajal/stemdl-data/test"

train_dataset = NPZDataset(train_data_dir)
test_dataset = NPZDataset(test_data_dir)
train_dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
test_dataloader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=True, num_workers=2)
```

A Simple CNN Model

```python
class CNN(nn.Module):
    def __init__(self, num_classes=231):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(in_channels=3, out_channels=12, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(in_channels=12, out_channels=24, kernel_size=3, stride=1, padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2)
        self.drop = nn.Dropout2d(p=0.2)
        self.fc = nn.Linear(in_features=32 * 32 * 24, out_features=num_classes)

    def forward(self, x):
        x = F.relu(self.pool(self.conv1(x)))
        x = F.relu(self.pool(self.conv2(x)))
        x = F.dropout(self.drop(x), training=self.training)
        x = x.view(-1, 32 * 32 * 24)
        x = self.fc(x)
        return torch.log_softmax(x, dim=1)
```

Optimizer and Loss Function

```
model = CNN()
print(model)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Training Loop

```
model = CNN()
print(model)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

model.train()
running_loss = 0.0
for epoch in range(1):
    for i, data in enumerate(train_dataloader):
        inputs, labels = data
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        if i % 10 == 1:
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / i:.3f}')
            running_loss = 0.0
```

Test Loop

```
def test(model, device, test_loader):
    # Switch the model to evaluation mode (so we don't backpropagate or drop)
    model.eval()
    test_loss = 0
    correct = 0
    with torch.no_grad():
        batch_count = 0
        for data, target in test_loader:
            batch_count += 1
            data, target = data.to(device), target.to(device)

            # Get the predicted classes for this batch
            output = model(data)

            # Calculate the loss for this batch
            test_loss += criterion(output, target).item()

            # Calculate the accuracy for this batch
            _, predicted = torch.max(output.data, 1)
            correct += torch.sum(target==predicted).item()

    # Calculate the average loss and total accuracy for this epoch
    avg_loss = test_loss / batch_count
    print('Validation set: Average loss: {:.6f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
        avg_loss, correct, len(test_loader.dataset),
        100. * correct / len(test_loader.dataset)))

    # return average loss for the epoch
    return avg_loss
```

The Demonstration

1

Running a CNN on CPU

2

Running the CNN on a GPU

3

Running the CNN on multiple GPUs

4

Running ResNet50 on multiple GPUs

5

Running on ResNet50 on multiple nodes

Code and Data

- Code: https://github.com/olcf/ai-training-series
- On Ascent: /gpfs/wolf/world-shared/trn018/sajal/ai-training-series/ai_at_scale
- Data: /gpfs/wolf/world-shared/trn018/sajal/data
- Login: ssh USERNAME@login1.ascent.olcf.ornl.gov