Introduction to Artificial Intelligence and Deep Learning for Science and Engineering

Zhao Zhang

Department of Electrical and Computer Engineering

Rutgers, the State University of New Jersey

RUTGERS

The Rutgers Artificial Intelligence and Data Science (RAD) Collaboratory

Outline

- Overview of scientific research methods
- National efforts to accommodate the rising AI need
- Introduction to artificial intelligence, machine learning, deep learning
- Hands-on with Hurricane Harvey Damage Assessment

The Progression of the Scientific Method

Increasing speed, automation, and scale

Empirical Science

1st Paradigm

Observation
Experimentation

Theoretical Science

2nd Paradigm

Scientific Laws in Physics, Chem, and others

Computational Science

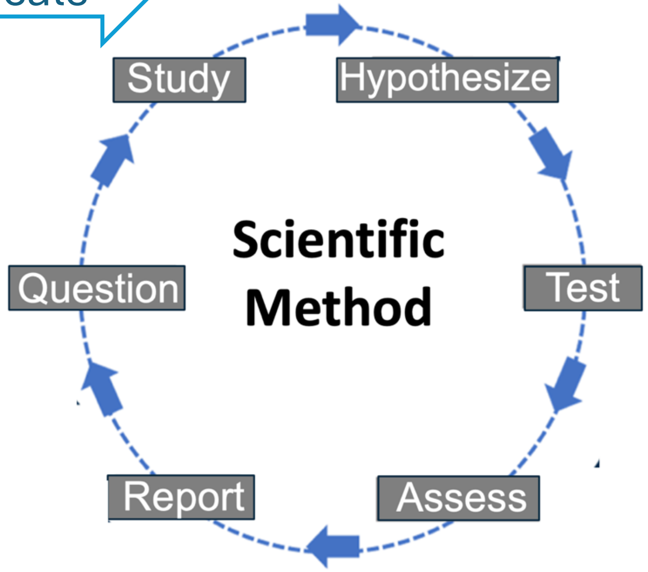
3rd Paradigm

Simulations
Molecular Dynamics
Mechanistic Models

Big Data-driven Science

4th Paradigm

Big data, machine learning
Patterns, anomalies
Visualization



Scientific knowledge at scale
AI-generated hypotheses
Autonomous testing

Credits: Ian T. Foster, UChicago

ML/DL in Science not So Long Ago

Data Analytics
- Classification
- Regression
- Clustering
- Dimensionality Reduction

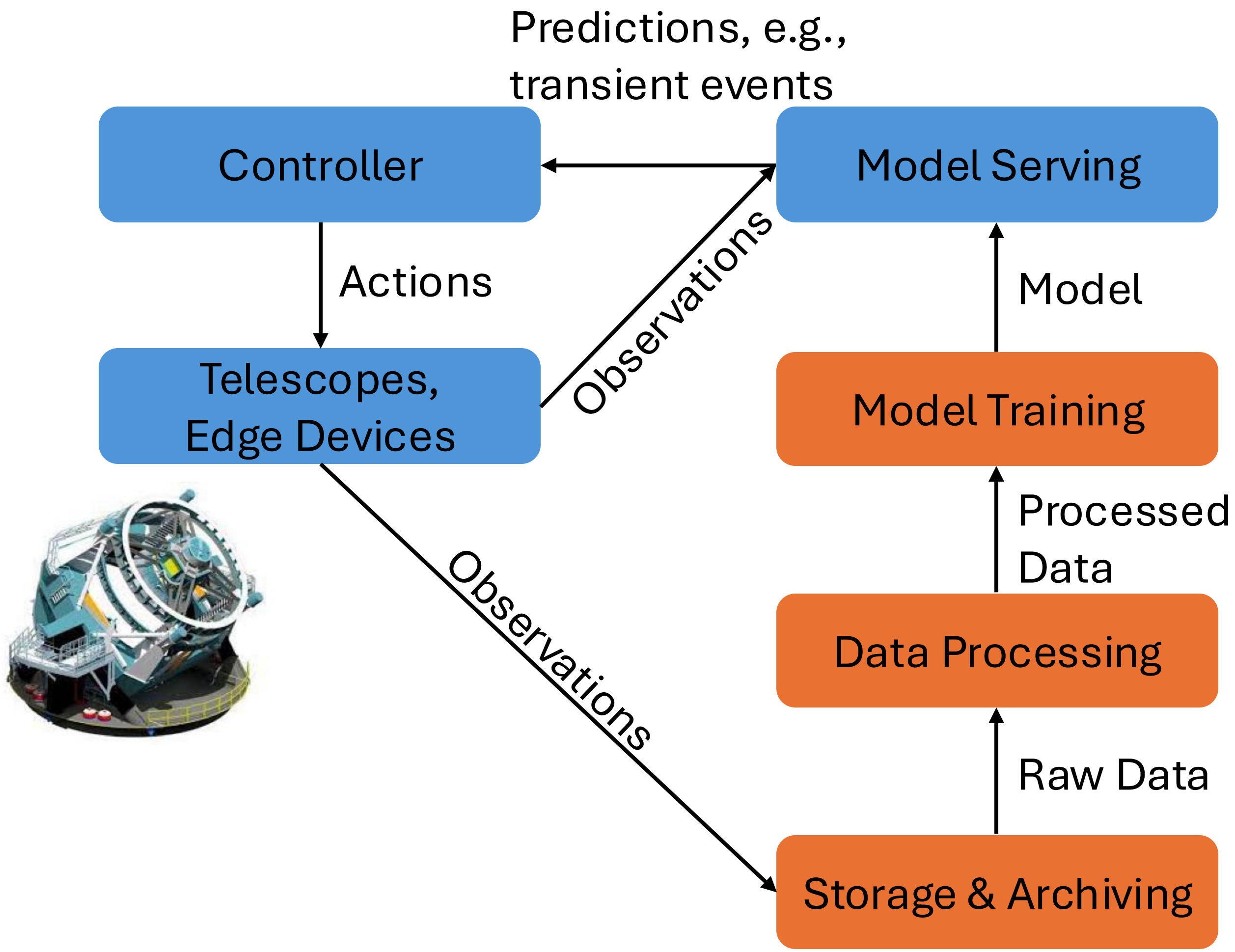
Inverse Problems
- Model Reconstruction
- Parameter Estimation
- Denoising

Surrogate Models
- Approximate Expensive Simulations
- Approximate Experiments
- Fill in Missing Models in Simulations

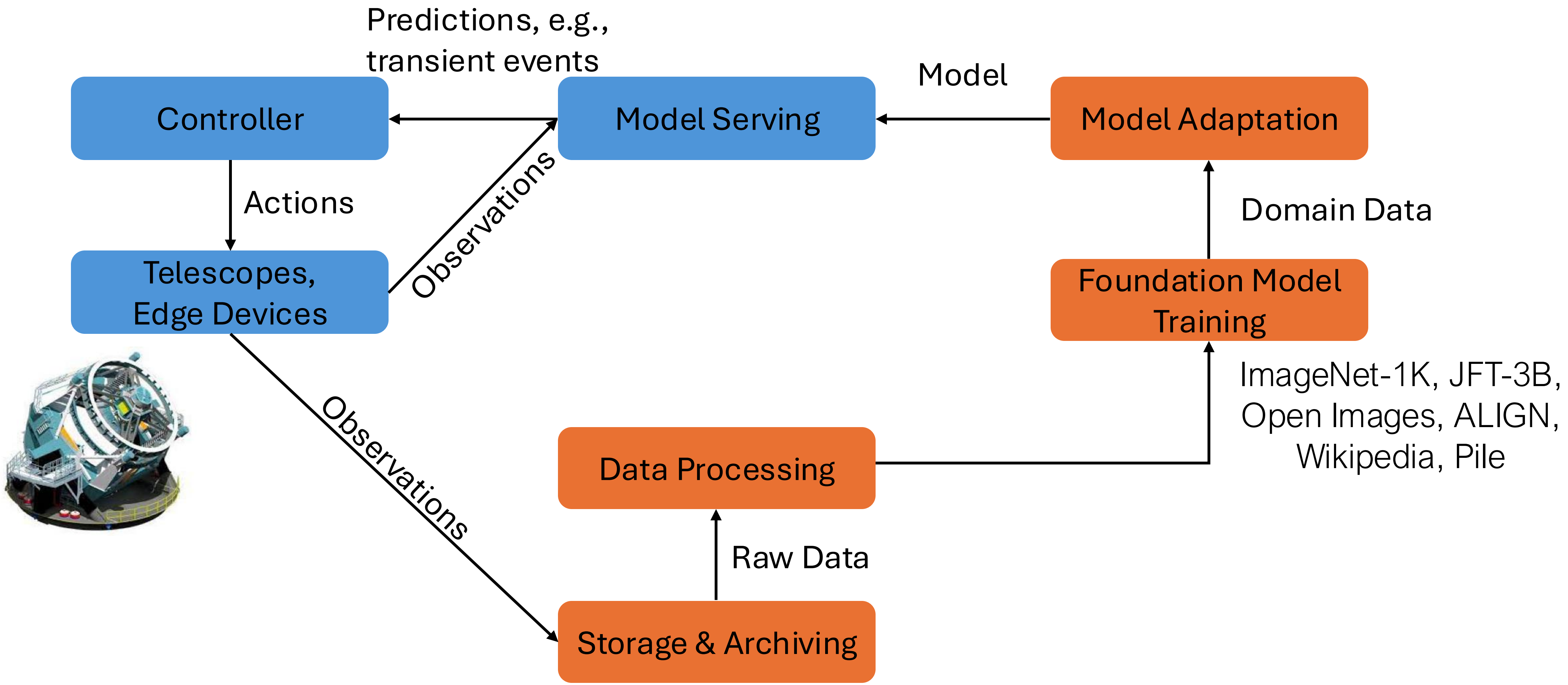
Design and Control
- Optimize Design of Experiments
- Control Instruments
- Navigate State Spaces
- Learn from Sparse Rewards

Credit: Kathy Yelick, in Monterey Data Conference, 2019

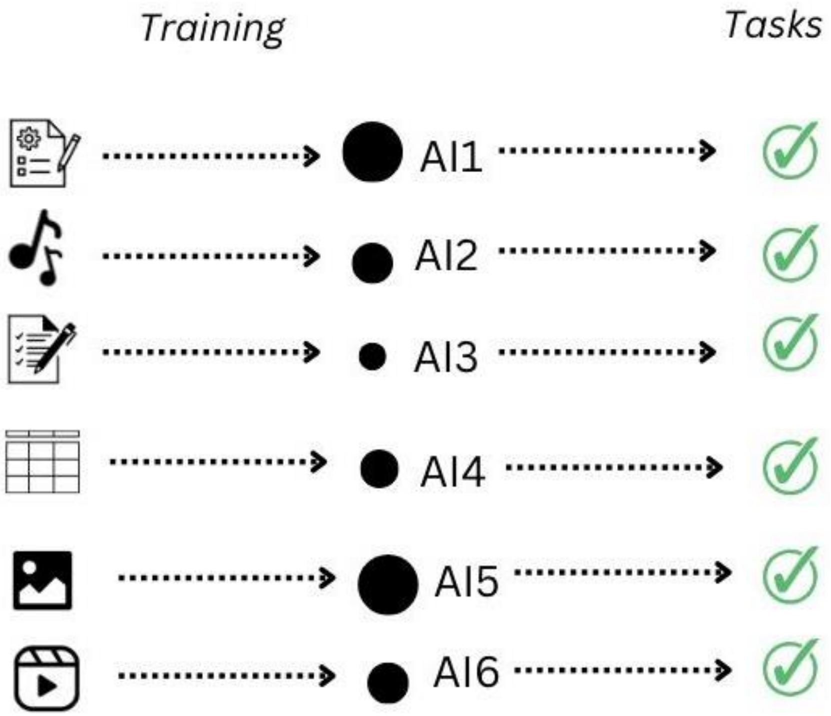
ML/DL in Science not So Long Ago



ML/DL in Science not So Long Ago

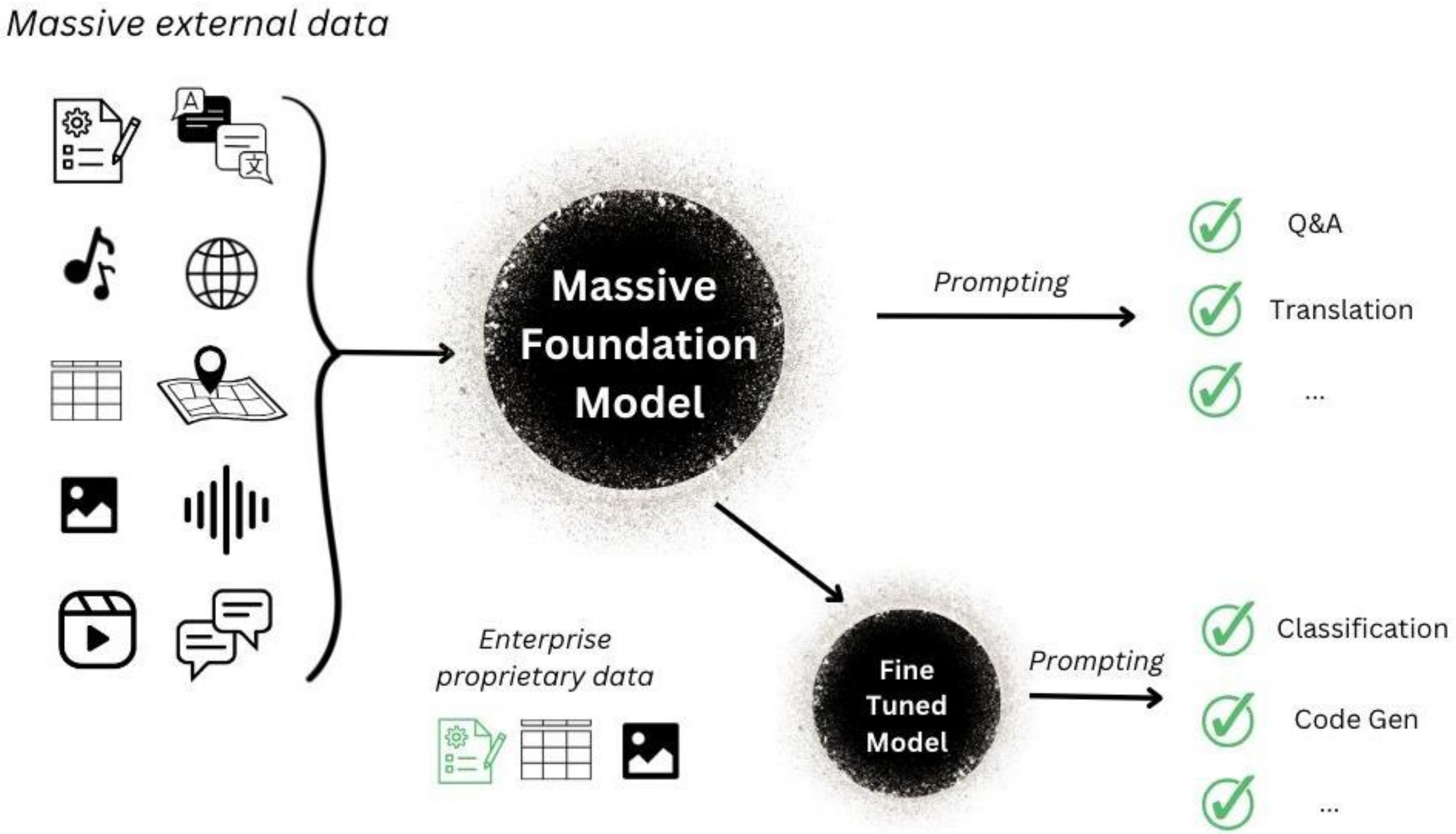


Traditional ML



- Individual siloed models
- Require task-specific training
- Lots of human supervised training

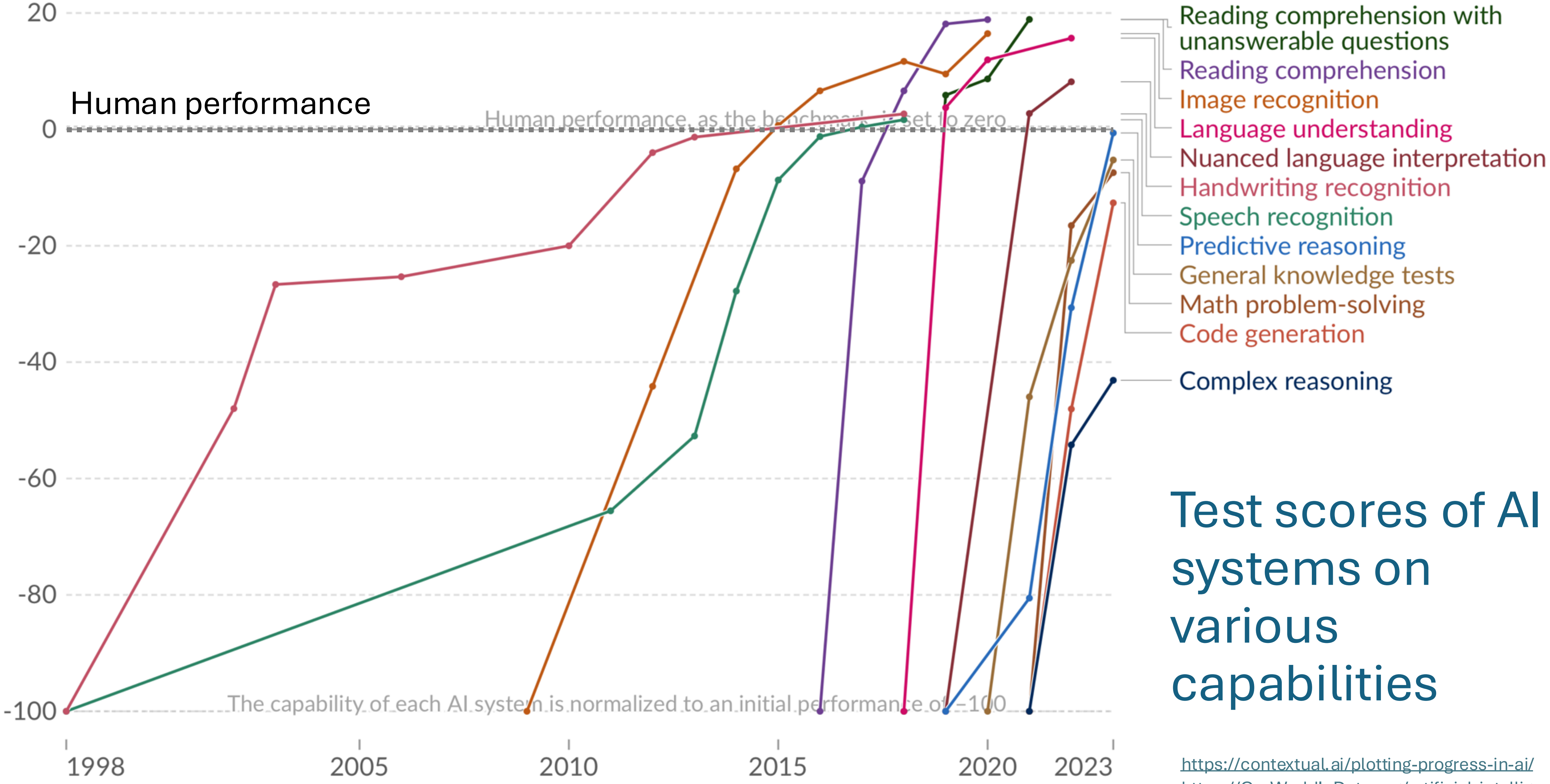
Foundation models



- Massive multi-tasking model
- Adaptable with little or no training
- Pre-trained unsupervised learning

Credits: Ian T. Foster, UChicago

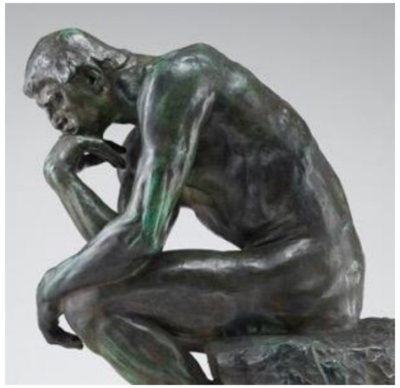
AI system capabilities are increasing rapidly

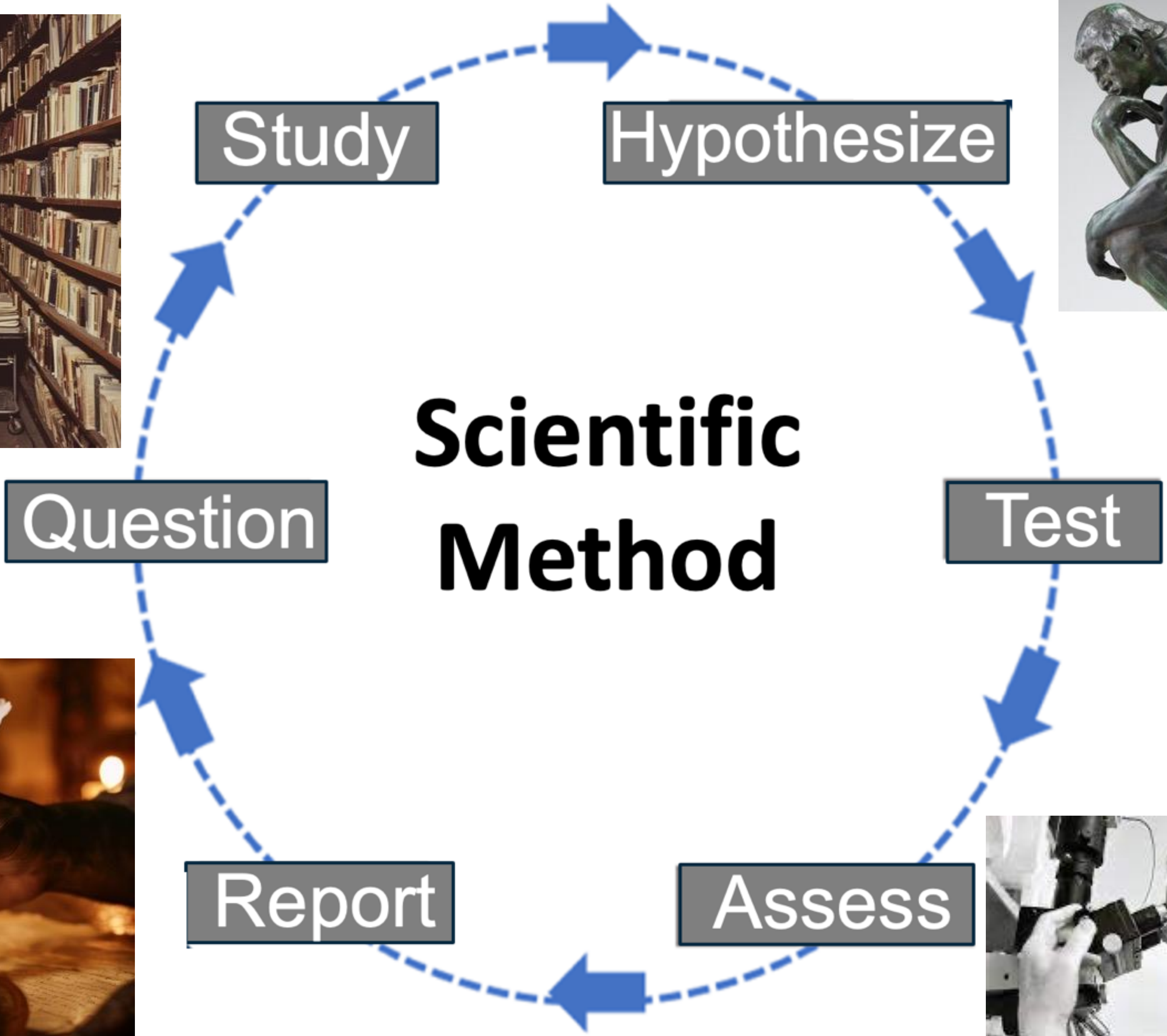


Test scores of AI systems on various capabilities

https://contextual.ai/plotting-progress-in-ai/
https://OurWorldInData.org/artificial-intelligence

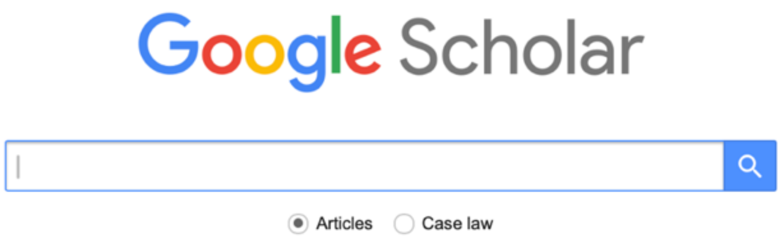
The scientific method remains slow and labor-intensive

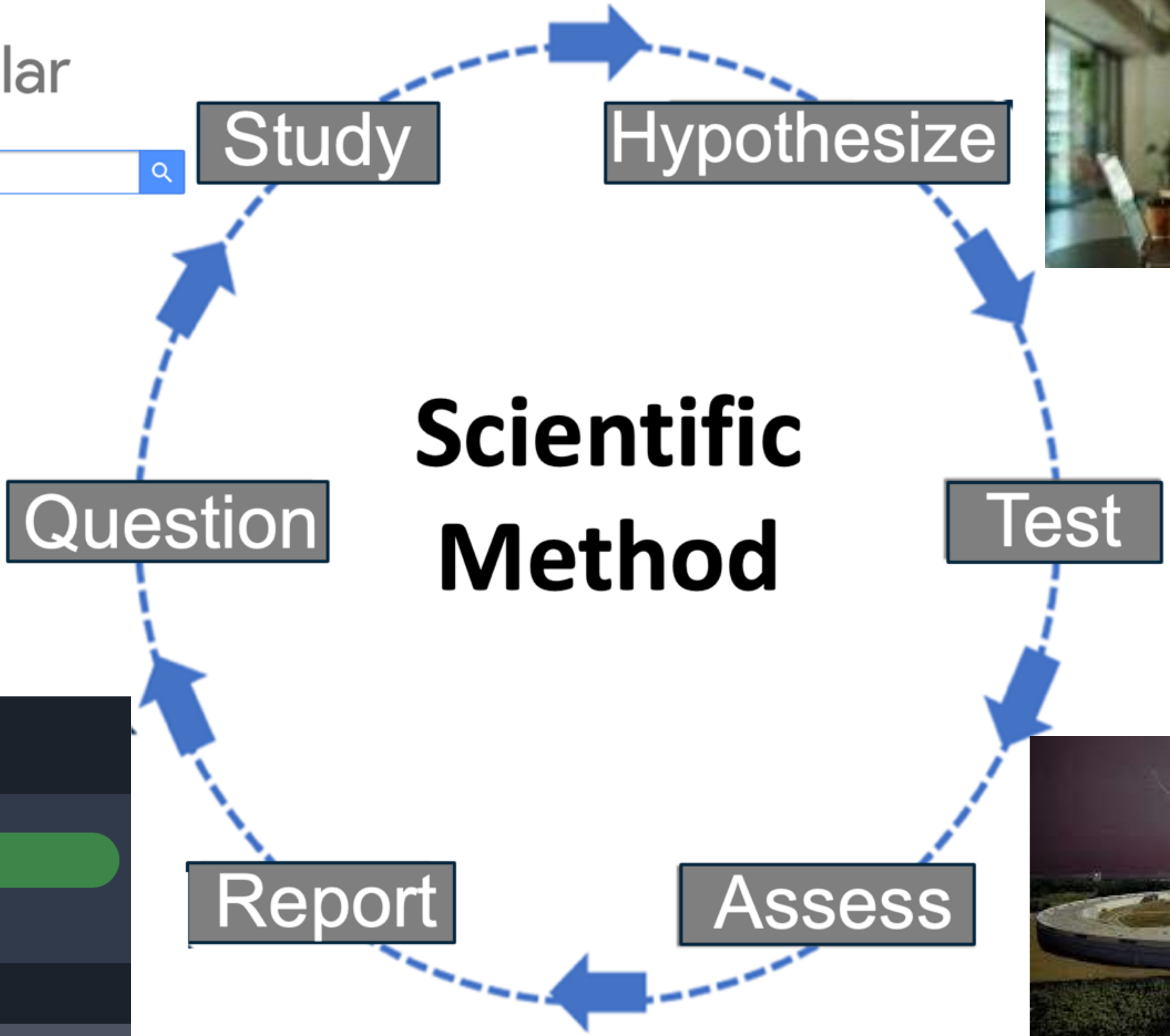


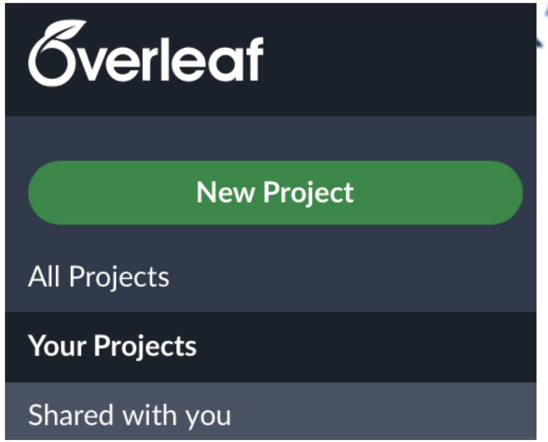


Credits: Ian T. Foster, UChicago

Despite acceleration of some steps via HPC etc.







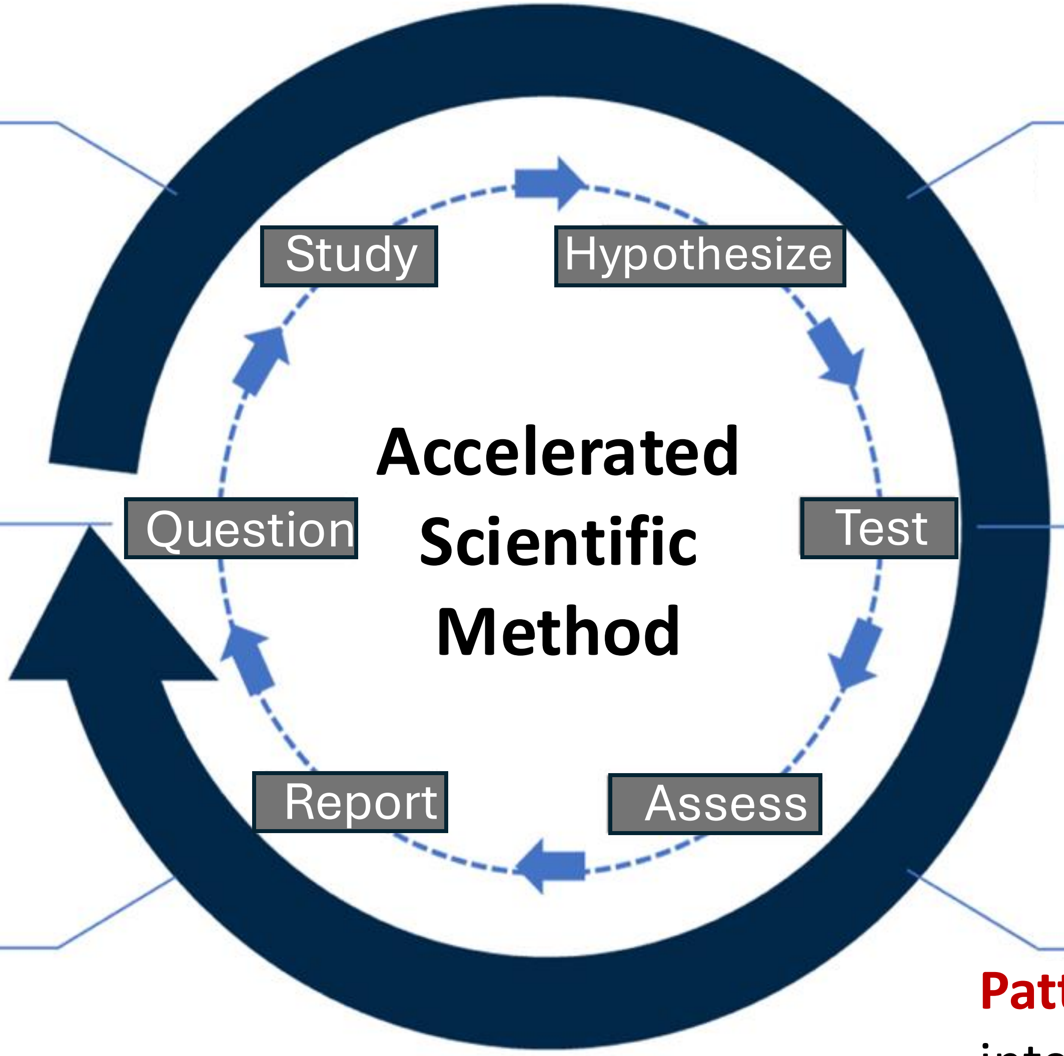
Credits: Ian T. Foster, UChicago

Engage AI assistants to help overcome bottlenecks

Extraction, integration and reasoning with knowledge at scale

Tools help **identify new questions** based on needs and gaps in knowledge

Machine representation of knowledge leads to new hypotheses and questions



Generative models automatically propose new hypotheses that expand discovery space

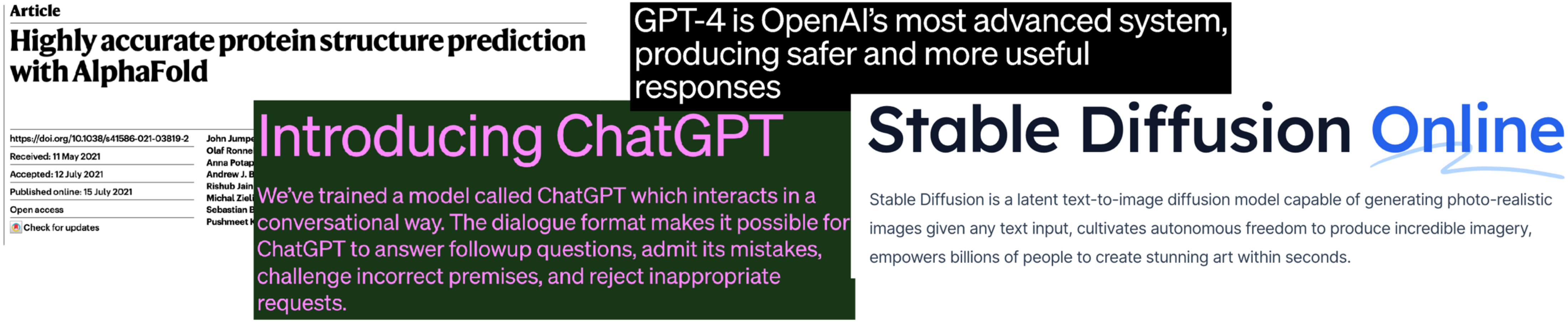
Robotic labs automate experimentation and bridge digital models and physical testing

Pattern and anomaly detection integrated with simulation and experiment extract new insights

https://doi.org/10.1038/s41524-022-00765-z

Credits: Ian T. Foster, UChicago

Foundation Model Training is Expensive



- Llama 3.1 405B takes 16,384 H100 GPUs for 2 months
- OPT-175B takes 1,024 A100 GPUs for 2 months
- OpenFold takes 128 A100 GPUs for 11 days
- GPT-NeoX 20B takes 96 A100 GPUs for 30 days
- Almost all popular large foundational models leverage transformers

- \$2.5k - \$50k (110 million parameter model)
- \$10k - \$200k (340 million parameter model)
- \$80k - \$1.6m (1.5 billion parameter model)

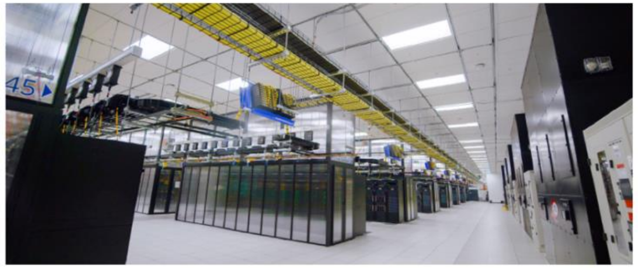
Sharir, Or, Barak Peleg, and Yoav Shoham. "The cost of training nlp models: A concise overview." arXiv preprint arXiv:2004.08900 (2020).

Industry Investment in AI Cyberinfrastructure

RESEARCH

Introducing the AI Research SuperCluster — Meta's cutting-edge AI supercomputer for AI research

RSC: Under the hood

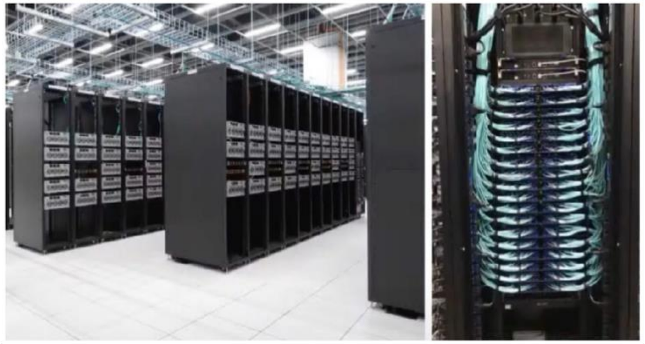


AI supercomputers are built by combining multiple GPUs into compute nodes, which are then connected by a high-performance network fabric to allow fast communication between those GPUs. RSC today comprises a total of 760 NVIDIA DGX A100 systems as its compute nodes, for a total of 6,080 GPUs — with each A100

Tesla Unveils Top AV Training Supercomputer Powered by NVIDIA A100 GPUs

'Incredible' GPU cluster powers AI development for Autopilot and full self-driving.

June 22, 2021 by DANNY SHAPIRO



Stability AI, the startup behind Stable Diffusion, raises $101M

Kyle Wiggers @kyle_l_wiggers / 12:01 PM CDT • October 17, 2022 Comment

Stability AI has a cluster of more than 4,000 Nvidia A100 GPUs running in AWS, which it uses to train AI systems, including Stable Diffusion. It's quite costly to maintain — Business Insider reports that Stability AI's operations and cloud expenditures exceeded $50 million. But Mostaque has repeatedly asserted that the company's R&D will enable it to train models more efficiently going forward.

Nvidia and Microsoft team up to build 'massive' AI supercomputer

/ The companies hope to create 'one of the most powerful AI supercomputers in the world,' capable of handling the growing demand for generative AI.

By JESS WEATHERBED
Nov 17, 2022, 6:44 AM CST | 2 Comments / 3 New

Meta's Llama 3.1 405B model was trained using over 16,000 NVIDIA H100 GPUs. This was the first Llama model to be trained at this scale.

Explanation

- The training process for Llama 3.1 405B required a large amount of computing power.
- Meta optimized their training infrastructure to handle the model's scale.
- The model was trained on over 15 trillion tokens.
- The training process took 54 days.

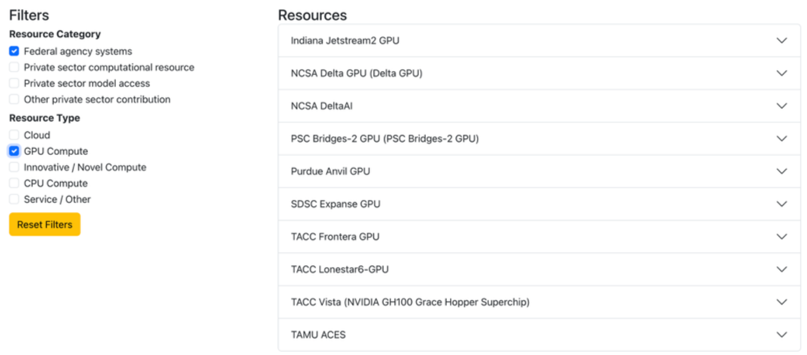
xAI Colossus is a supercomputer built by xAI, a company founded by Elon Musk, to train and power the AI chatbot Grok. It's located in Memphis, Tennessee, in a former Electrolux manufacturing plant.

Features:

- **GPUs**: The supercomputer has over 100,000 Nvidia H100 GPUs, which are some of the most powerful processing chips available
- **Liquid cooling**: The GPUs are liquid-cooled
- **Networking**: The supercomputer uses Nvidia Spectrum-X Ethernet networking
- **Storage**: The supercomputer has exabytes of storage

National Investment in AI Cyberinfrastructure

- To accommodate the increasing need of HPC for AI, the US government has heavily invested in supercomputers:
  - TACC Horizon, O(1000) GPUs, to deploy in 2026, funded by NSF LCCF
  - NERSC Perlmutter, +7,000 Nvidia A100s, deployed in 2021
  - ALCF Polaris, +2,000 NVIDIA A100s, deployed in 2022
  - OLCF Frontier, 37,888 AMD MI250X GPUs, deployed in 2021
  - ALCF Aurora, 63,744 Intel GPU Max Series, deployed in 2023



National Investment in AI Cyberinfrastructure

The National Artificial Intelligence Research Resource (NAIRR) Pilot

The NAIRR Pilot aims to connect U.S. researchers and educators to computational, data, and training resources needed to advance AI research and research that employs AI. Federal agencies are collaborating with government-supported and non-governmental partners to implement the Pilot as a preparatory step toward an eventual full NAIRR implementation.

Operational focus areas

NAIRR Open

This focus area, led by NSF, will support open AI research by providing access to diverse AI resources via the NAIRR Pilot Portal and coordinated allocations.

NAIRR Secure

This focus area, co-led by the National Institutes of Health and the Department of Energy, will support AI research requiring privacy and security-preserving resources and assemble exemplar privacy-preserving resources.

NAIRR Software

This focus area, led by NSF, will facilitate and investigate interoperable use of AI software, platforms, tools and services for NAIRR pilot resources.

NAIRR Classroom

This focus area, led by NSF, will reach new communities through education, training, user support and outreach.

Filters

Resource Category

- [x] Federal agency systems
- [ ] Private sector computational resource
- [ ] Private sector model access
- [ ] Other private sector contribution

Resource Type

- [ ] Cloud
- [x] GPU Compute
- [ ] Innovative / Novel Compute
- [ ] CPU Compute
- [ ] Service / Other

Reset Filters

Resources

- Indiana Jetstream2 GPU
- NCSA Delta GPU (Delta GPU)
- NCSA DeltaAI
- PSC Bridges-2 GPU (PSC Bridges-2 GPU)
- Purdue Anvil GPU
- SDSC Expanse GPU
- TACC Frontera GPU
- TACC Lonestar6-GPU
- TACC Vista (NVIDIA GH100 Grace Hopper Superchip)
- TAMU ACES

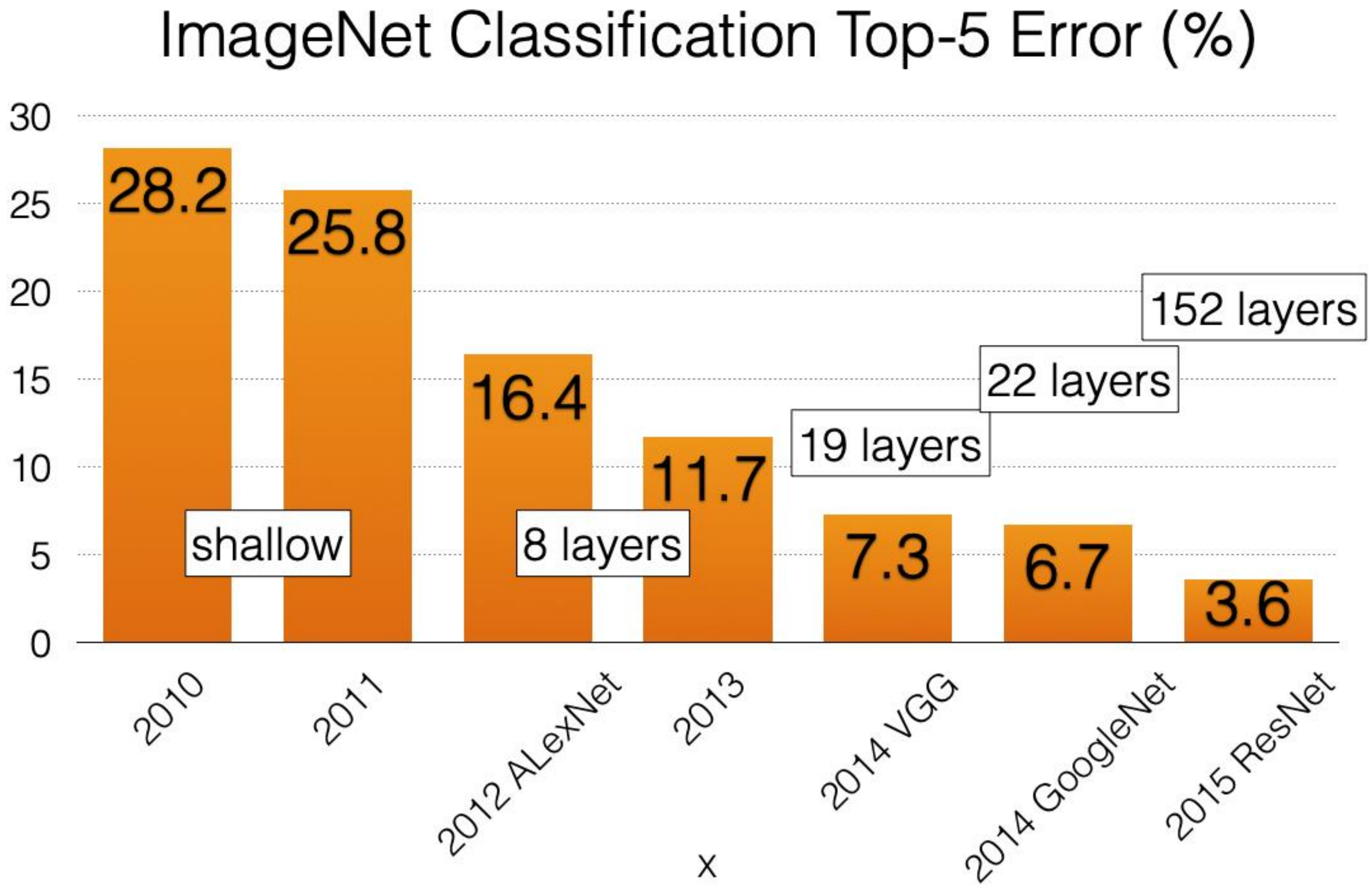
A Quick Overview of Deep Learning

- 1960s — Cybernetics
- 1990s — Connectionism + Neural Networks
- 2010s — Deep Learning

- Two key factors for the on-going renaissance
  - Computing capability
  - Data

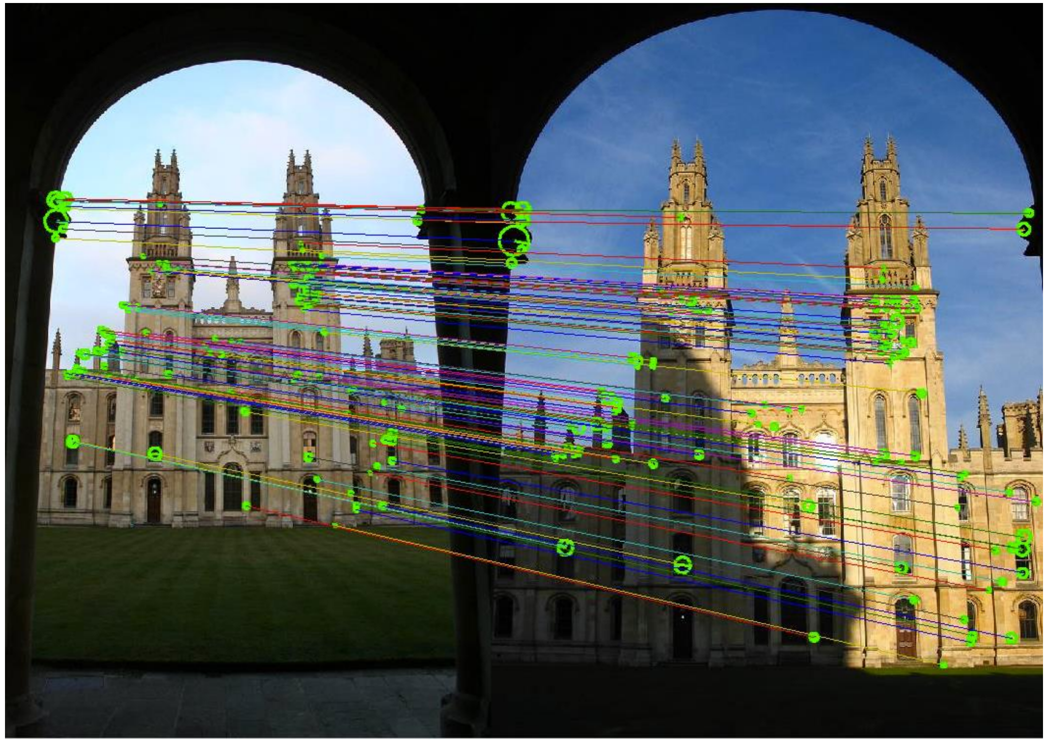
A Quick Overview of Deep Learning

- Image Classification with ImageNet Dataset



From Classical ML to DL

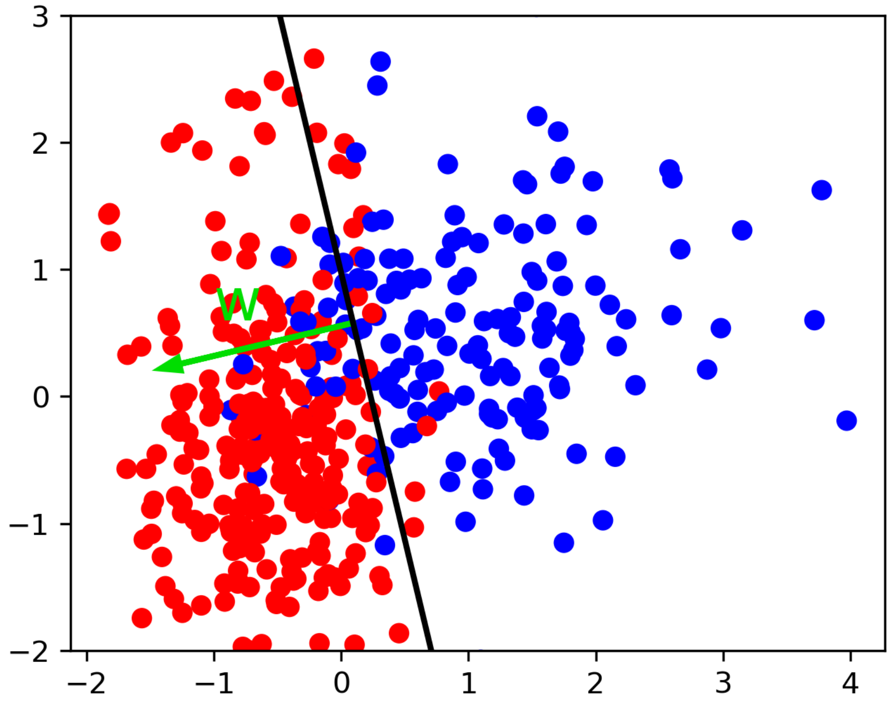
Feature Engineering



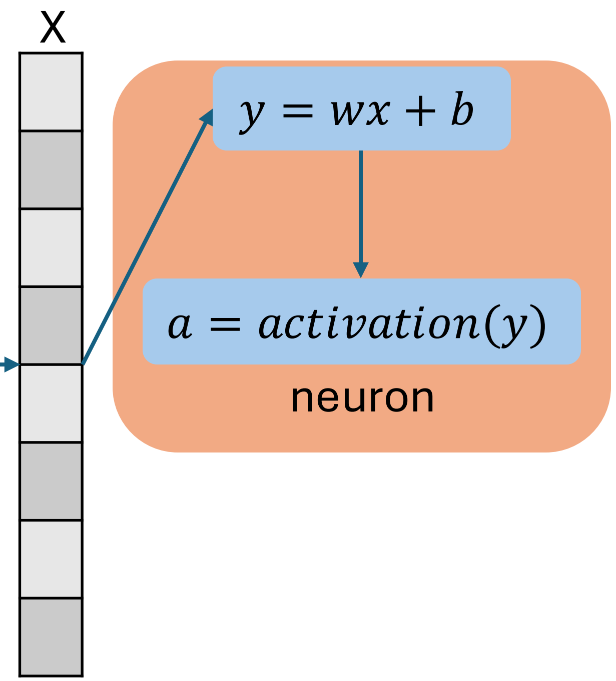
Linear Regression:

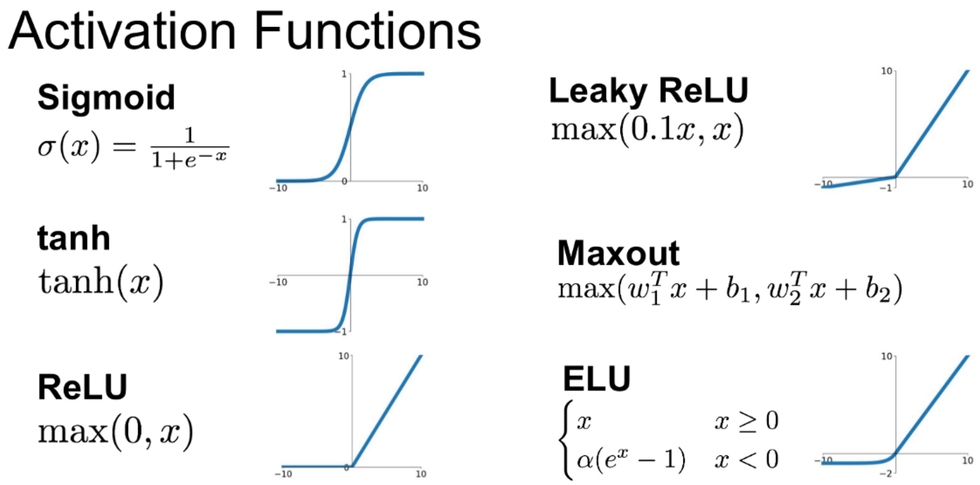
$$y = wx + b,$$

$$Loss = \sum_{i=1}^{N} (wx^i + b - y'^i)^2$$

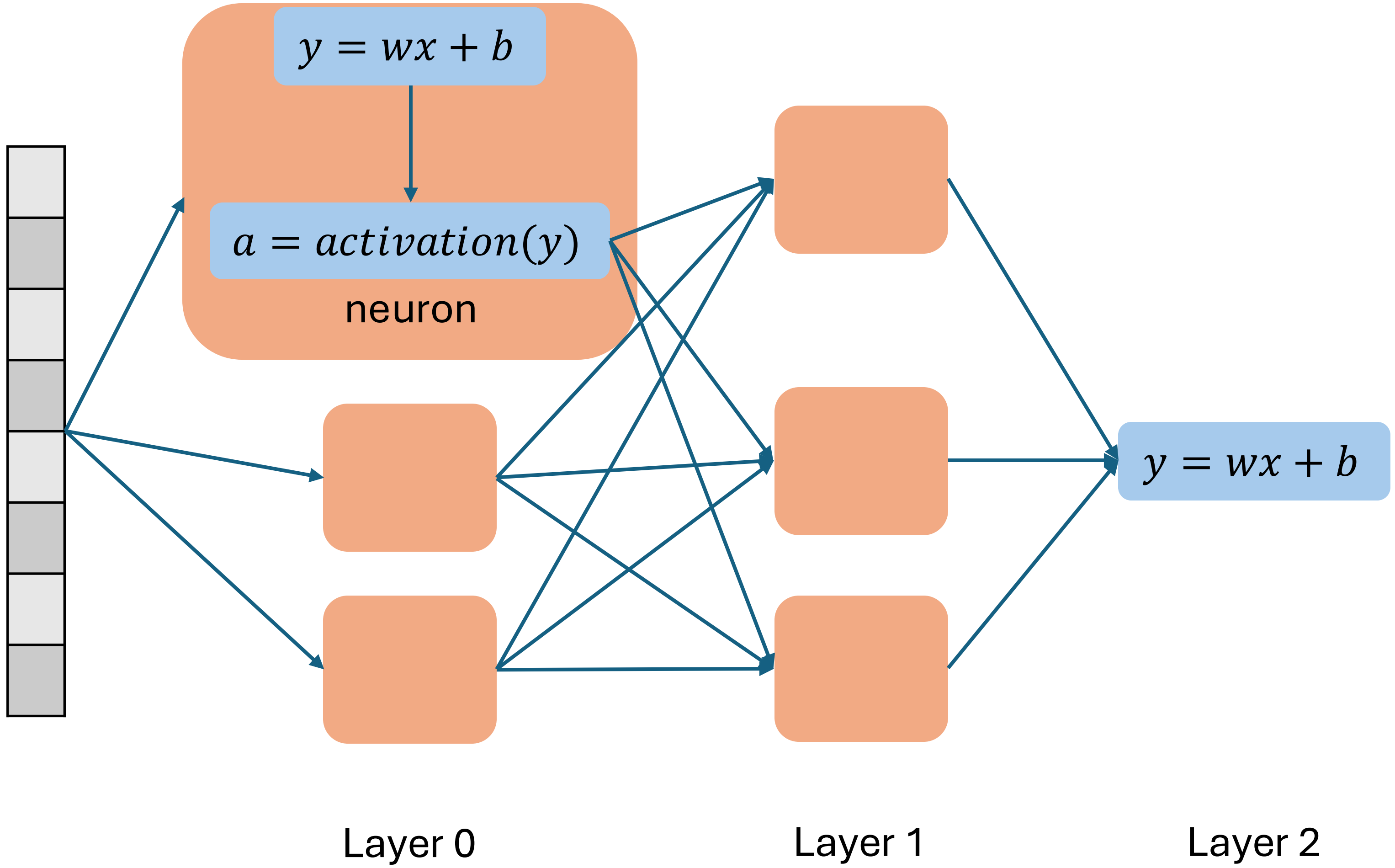


From Linear Regression to Neural Networks



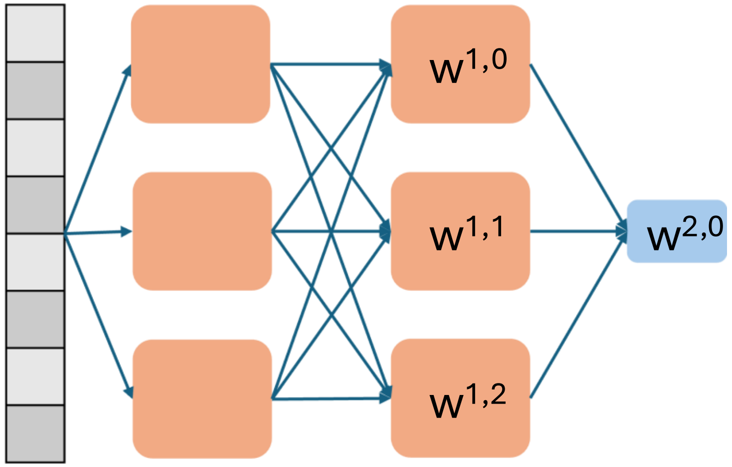


From Linear Regression to Neural Networks



From Linear Regression to Neural Networks

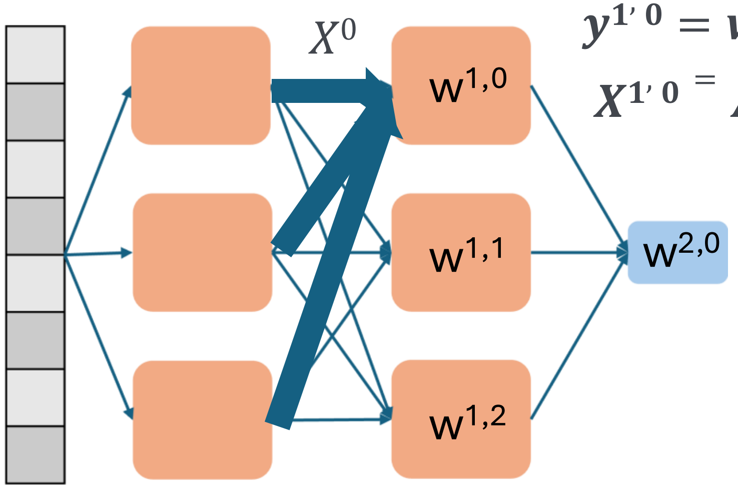
- Now we have labeled data
- We can calculate y and the error with label y'
- We can then update $w^{2,0}$
- How can we update $w^{1,0}$, $w^{1,1}$, $w^{1,2}$?



From Linear Regression to Neural Networks

- The back-propagation algoirithm
  - $W^{1'0} = W^{1'0} - \lambda * \partial Loss/\partial W^{1'0}$
  - $\partial Loss/\partial W^{1'0} = \partial Loss/\partial y^{2'0} * \partial y^{2'0}/\partial Activate^{1'0} * \partial Activate^{1'0}/\partial y^{1'0} * \partial y^{1'0}/\partial W^{1'0}$

4 3 2 1



$\boldsymbol{y^{1'0}} = \boldsymbol{w^{1'0}} * X^0 + b^{1'0}$ 1

$\boldsymbol{X^{1'0}} = \boldsymbol{Activate}(\boldsymbol{y^{1'0}})$ 2

$\boldsymbol{y^{2'0}} = w^{2'0} * \boldsymbol{X1} + b^{2'0}$ 3

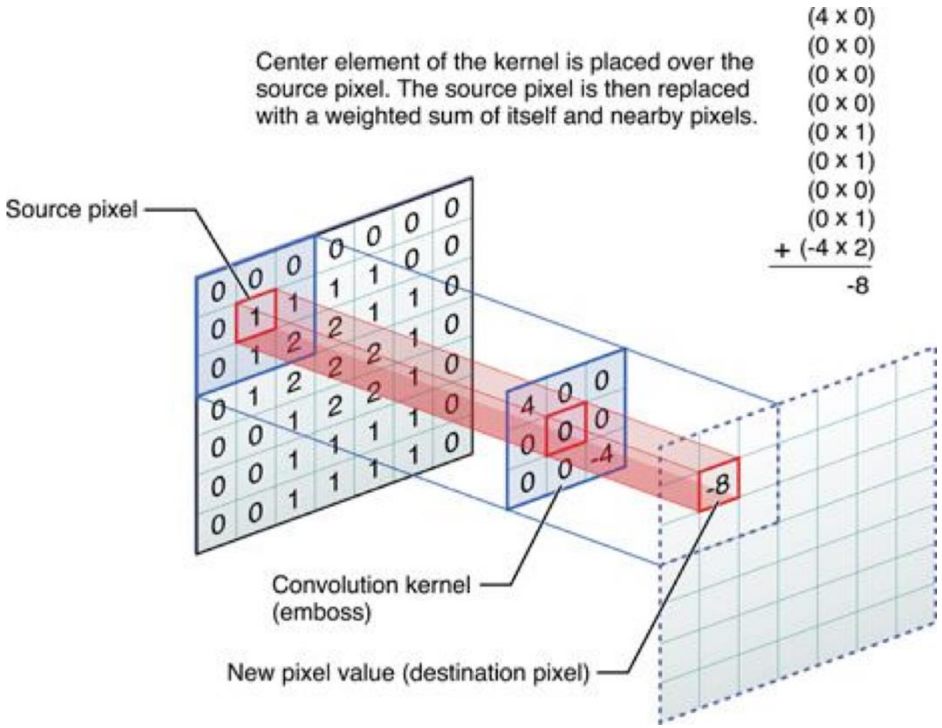
$\boldsymbol{Loss} = 1/2 * (\boldsymbol{y^{2'0}} - y')2$ 4

From Linear Regression to Neural Networks

- Stochastic Gradient Descent
  - Divides a labeled training dataset into two parts. E.g., 80% and 20%, referred as training and validation dataset, respectively
  - Trains a neural network iteratively
    - Takes a mini-batch of data, e.g., 64 items out of 2,048
    - An epoch is 2048/64=32 iterations/steps
  - Validates the model with validation dataset
    - Monitors the training loss and validation metrics, e.g., training/validation accuracy
- How many epochs is enough?

Convolutional Neural Network

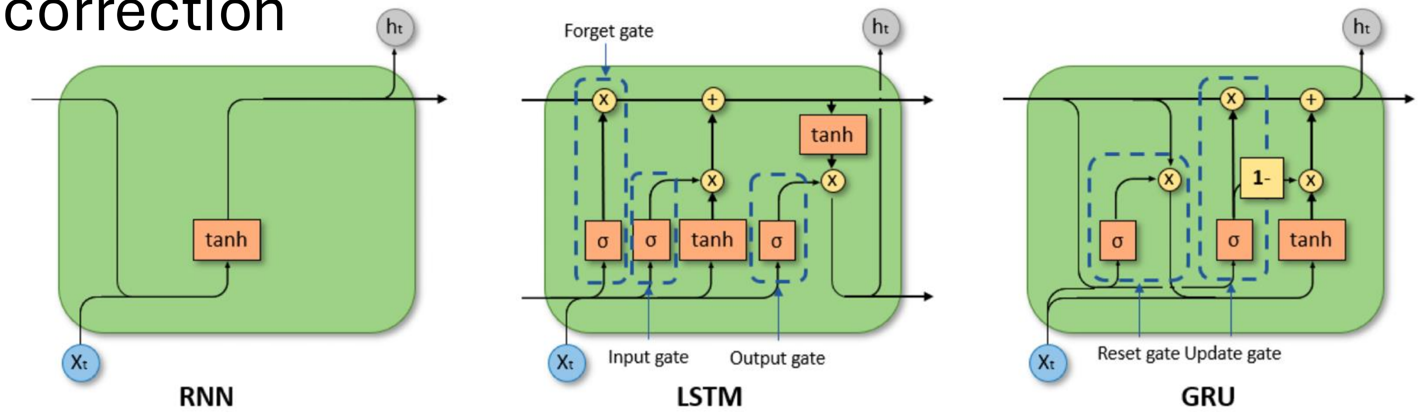
- What we just saw is a multi-layer perceptron (MLP) network
- If in any layer, there is a convolution operations, it is called convolutional neural network
  - Often coupled with pooling operation
- Example applications:
  - Image classification
  - Object detection
  - Autonomous driving



https://ikhlestov.github.io/pages/machine-learning/convolutions-types/

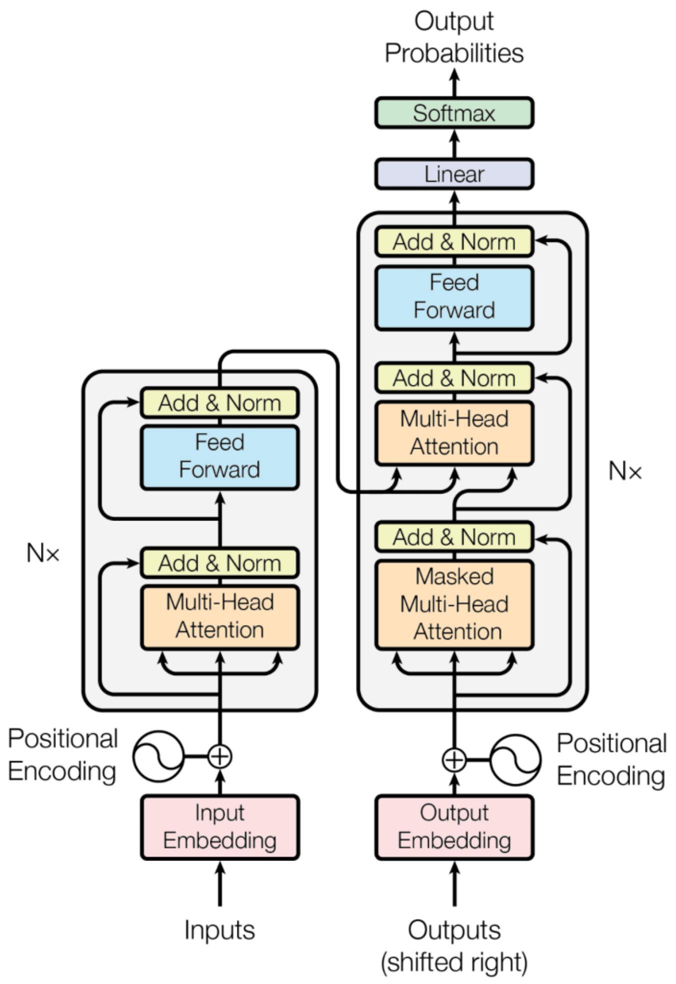
Recurrent Neural Network

- Recurrent Neural Network is another typical neural network architecture, mainly used for ordered/sequence input
- RNNs provide a way of use information about $X_{t-i}, \ldots, X_{t-1}$ for inferring X_t
- Example applications:
  - Language models, i.e. auto correction
  - Machine Translation
  - Auto image captioning
  - Speech Recognition
  - Autogenerating Music

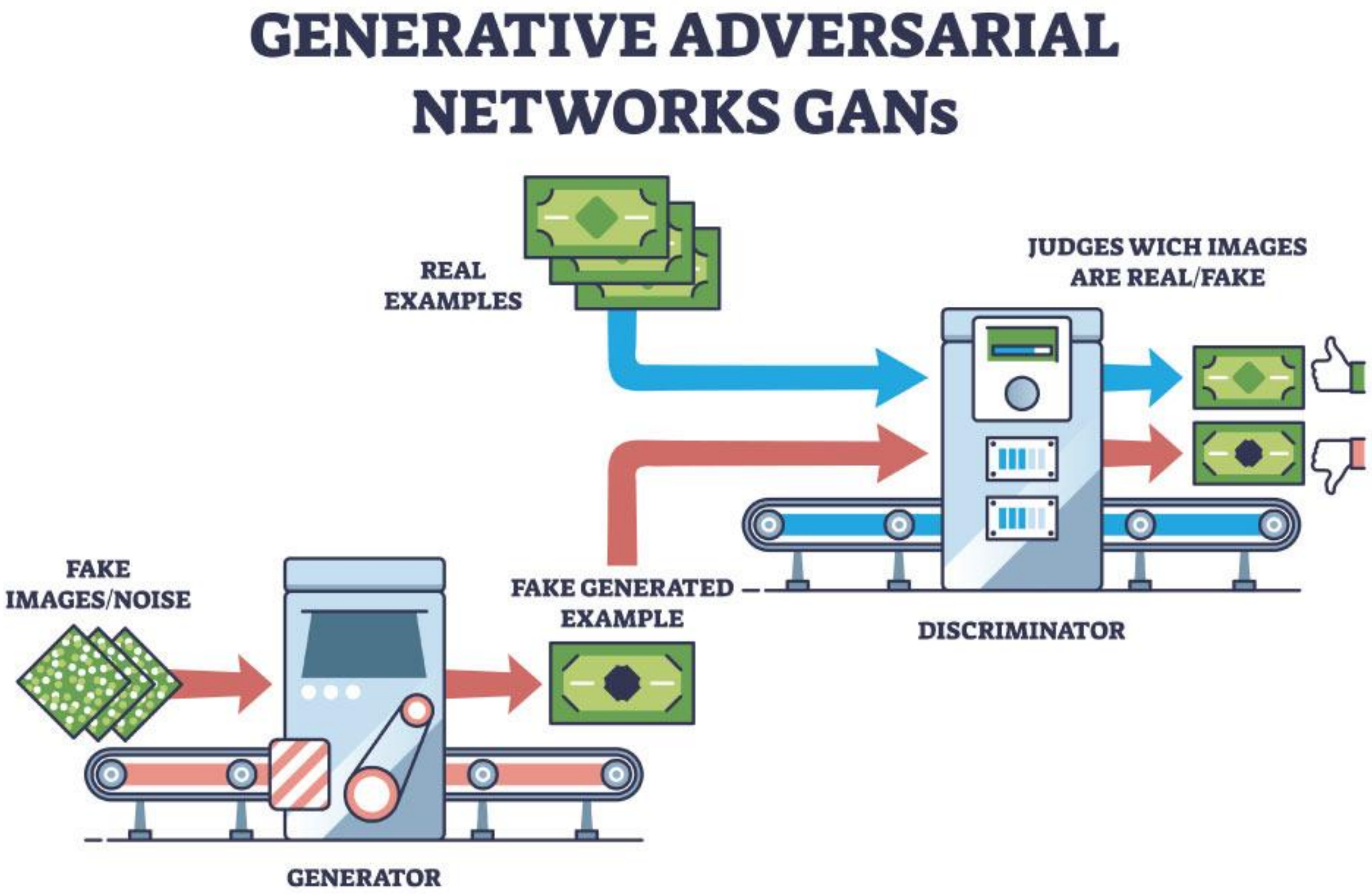


Transformer Network

- State-of-the-art operator
- Proposed by Google
- Attention Mechanism
- Fundamental in Large Language Models, e.g., BERT, GPT-3, chatGPT, Vision Transformer, AlphaFold

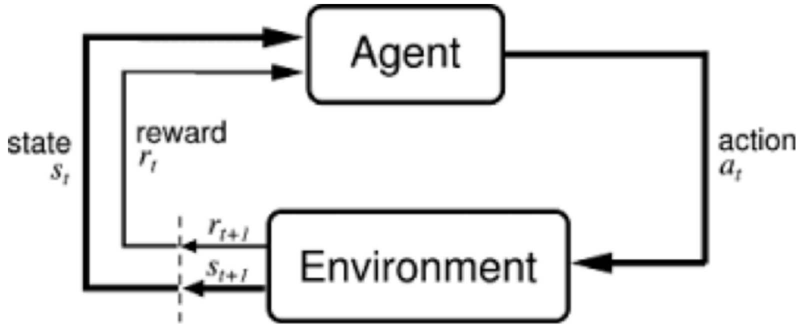


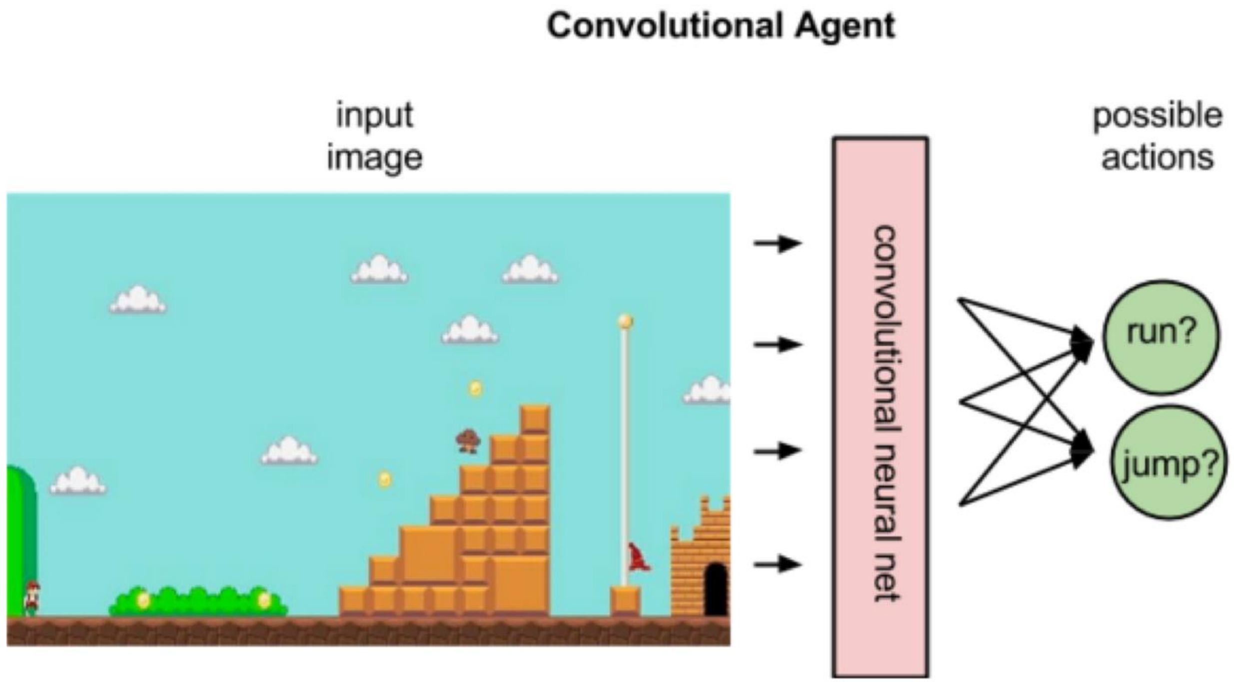
Generative Adversarial Network



https://pg-p.ctme.caltech.edu/blog/ai-ml/what-is-generative-adversarial-network-types

Deep Reinforcement Learning





https://skymind.ai/wiki/deep-reinforcement-learning

Notion Recap

- Neural Network Architecture
  - Multi-layer Perceptron
  - Convolutional Neural Network
  - Recurrent Neural Network
  - Transformer Network
- Activation, Loss, and Optimization
  - Activation Function
  - Loss Function
  - Back-propagation
  - Gradient Descent
  - Stochastic Gradient Descent
- Training and Validating
  - Training Dataset
  - Validation/Test Dataset
  - Training Accuracy
  - Validation/Test Accuracy Training Loss
  - Validation/Test Loss
  - Epoch
  - Iteration/Step

Deep Learning Software Stack

<table>
<tr><td>Programming</td><td colspan="12">PyTorch</td><td colspan="12">Torch Lightning</td><td colspan="12">JAX</td><td colspan="12">TensorFlow</td><td colspan="12">MXNet</td></tr>
<tr><td>Distributed</td><td colspan="12">DeepSpeed</td><td colspan="12">torch.distrib uted</td><td colspan="12">torch.FSDP</td><td colspan="12">Accelerate</td><td colspan="12">ZeRO</td></tr>
<tr><td>Resource Management</td><td colspan="30">Slurm</td><td colspan="30">Kubernetes</td></tr>
<tr><td>Communication</td><td colspan="20">NCCL</td><td colspan="20">MPI</td><td colspan="20">Gloo</td></tr>
<tr><td>Interconnect</td><td colspan="15">NVLink</td><td colspan="15">Slingshot</td><td colspan="15">Infiniband</td><td colspan="15">RoCE</td></tr>
</table>

PyTorch

- You can compose a PyTorch program in four steps:
  - Dataset Preparation
  - Model Definition
  - Optimizer Specification
  - Training Instrumentation

PyTorch Dataset

- Dataset —> Dataloader
- PyTorch has built-in datasets, e.g., CIFAR10

```
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets
from torchvision.transforms import ToTensor, Lambda, Compose

train_data = datasets.CIFAR10(root="/tmp", train=True, download=True, transform=ToTensor())

test_data = datasets.CIFAR10(root="/tmp", train=False, download=True, transform=ToTensor())

batch_size = 128

# Create data loaders.
train_dataloader = DataLoader(train_data, batch_size=batch_size)
test_dataloader = DataLoader(test_data, batch_size=batch_size)
```

PyTorch Model

- Inherits nn.Module

```
import torch
from torch import nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model = Net().to(device)
```

PyTorch Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)
```

PyTorch Training

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

PyTorch Training

```
def train(dataloader, model, loss_fn, optimizer):
    size = len(dataloader.dataset)
    for batch, (X, y) in enumerate(dataloader):
        X, y = X.to(device), y.to(device)

        # Compute prediction error
        pred = model(X)
        loss = loss_fn(pred, y)

        # Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if batch % 100 == 0:
            loss, current = loss.item(), batch * len(X)
            print(f"loss: {loss:>7f}  [{current:>5d}/{size:>5d}]")
```

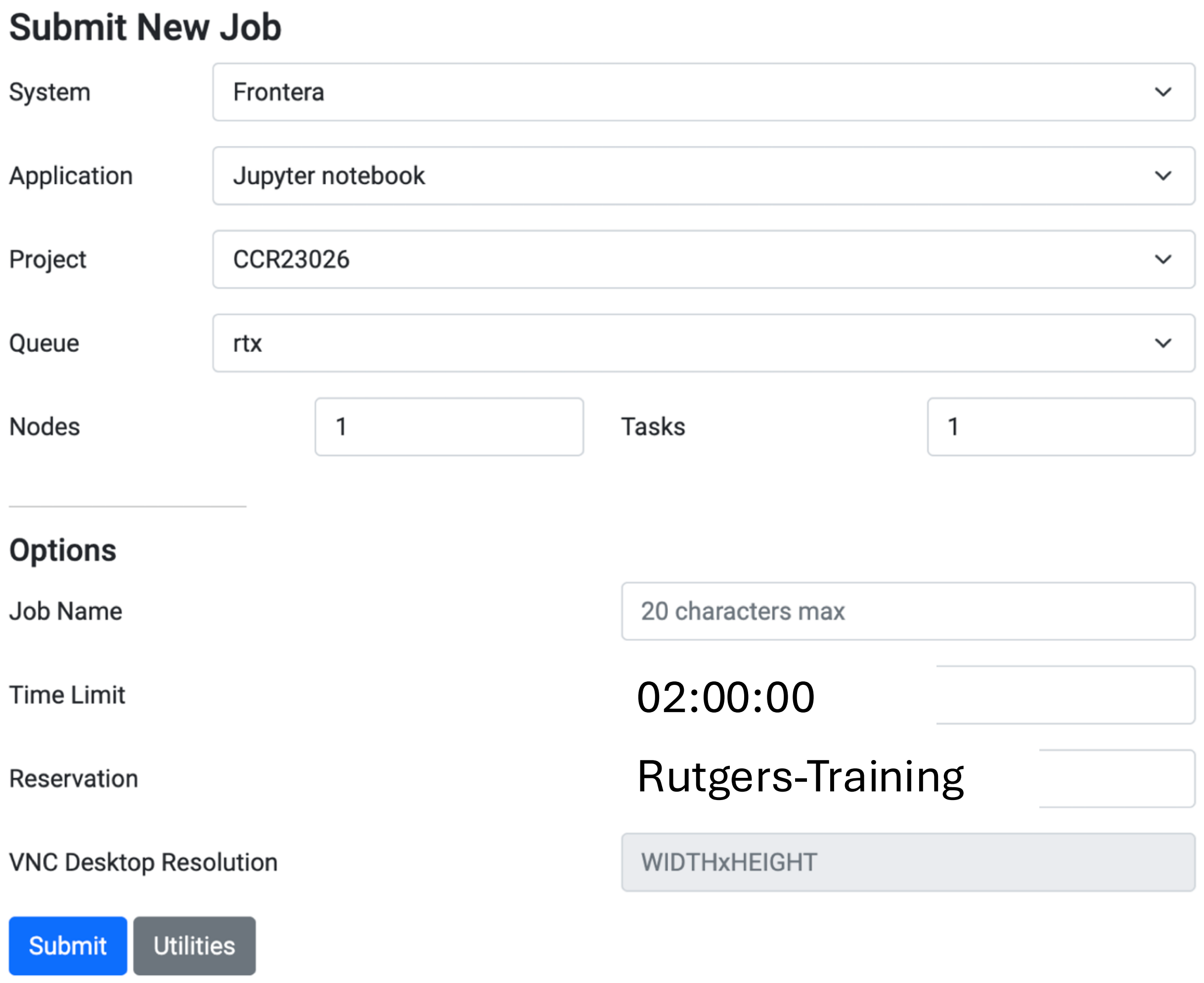
```
def test(dataloader, model, loss_fn):
    size = len(dataloader.dataset)
    num_batches = len(dataloader)
    model.eval()
    test_loss, correct = 0, 0
    with torch.no_grad():
        for X, y in dataloader:
            X, y = X.to(device), y.to(device)
            pred = model(X)
            test_loss += loss_fn(pred, y).item()
            correct += (pred.argmax(1) ==
                        y).type(torch.float).sum().item()
    test_loss /= num_batches
    correct /= size
    print(f"Test Error: \n Accuracy: {(100*correct):>0.1f}%,
          Avg loss: {test_loss:>8f} \n")
```

Hands-on Exercise

- ssh username@frontera.tacc.utexas.edu
  - cp -r /home1/00946/zzhang/RAD-tutorial ~/
  - source ~/RAD-tutorial/env.sh

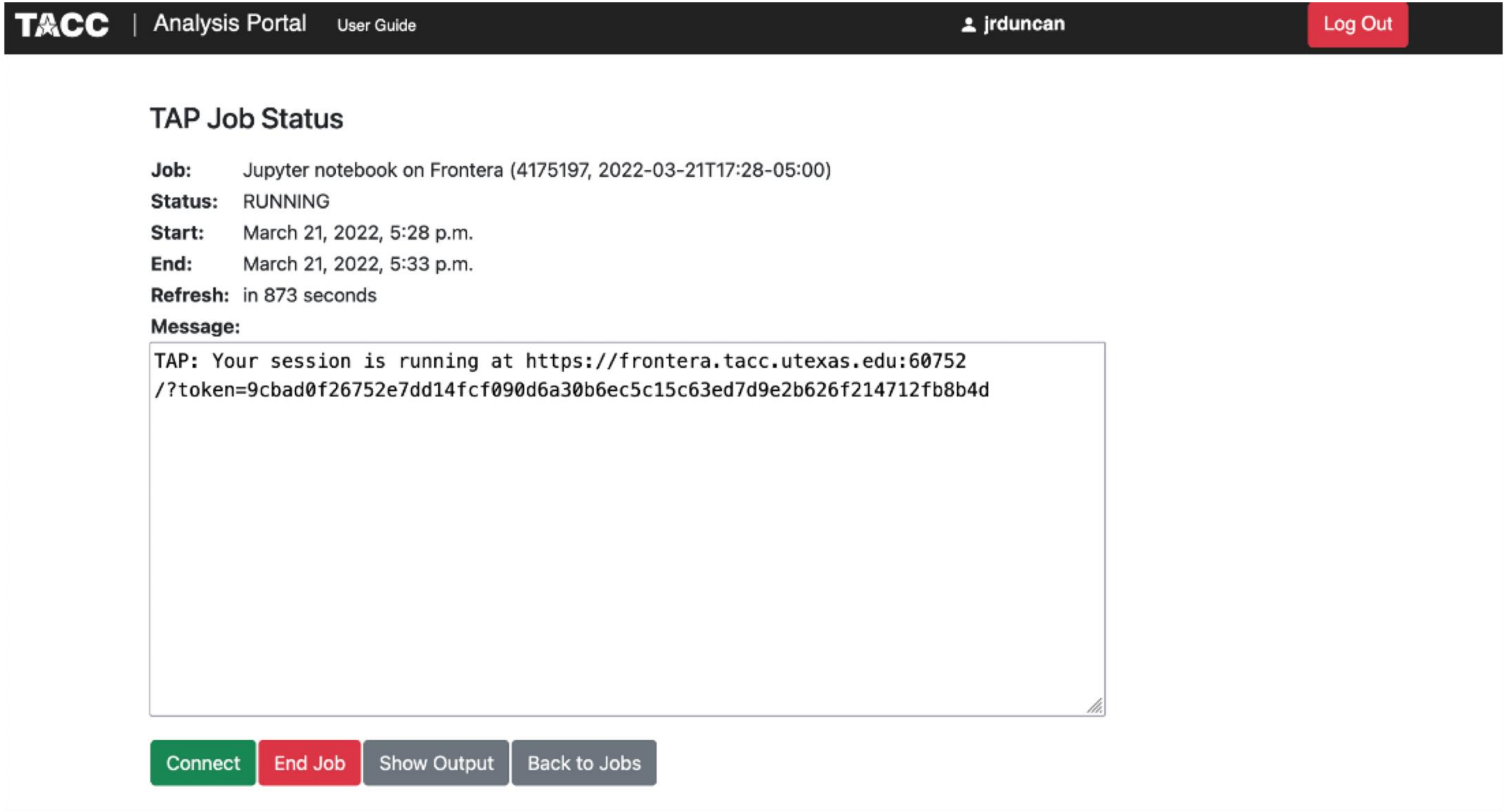
Hands-on Exercise

- https://tap.tacc.utexas.edu/



Rutgers_RTX

Hands-on Exercise



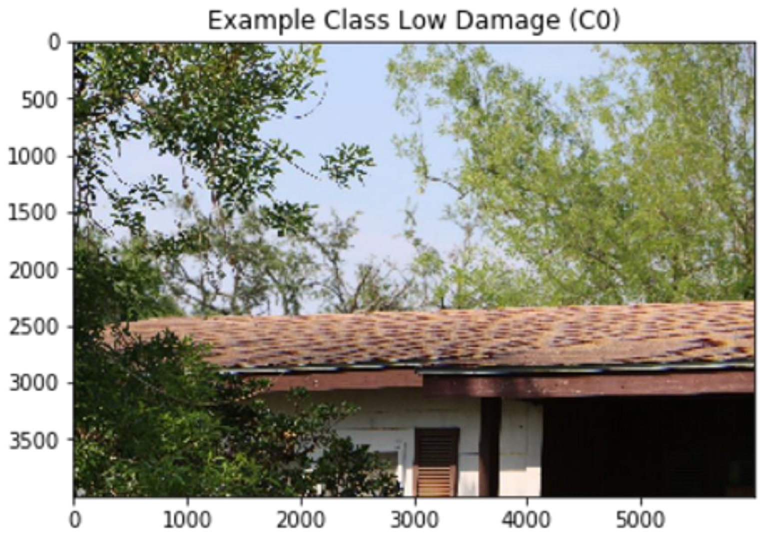
Hands-on Exercise

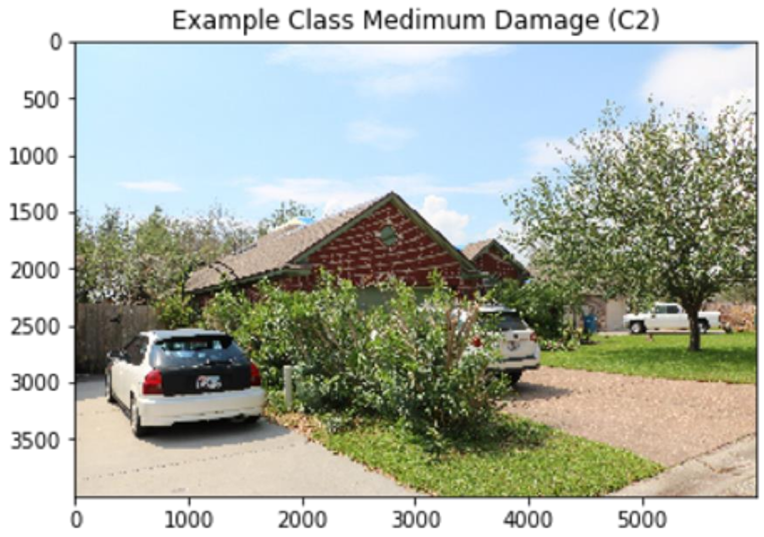
Hands-on Exercise

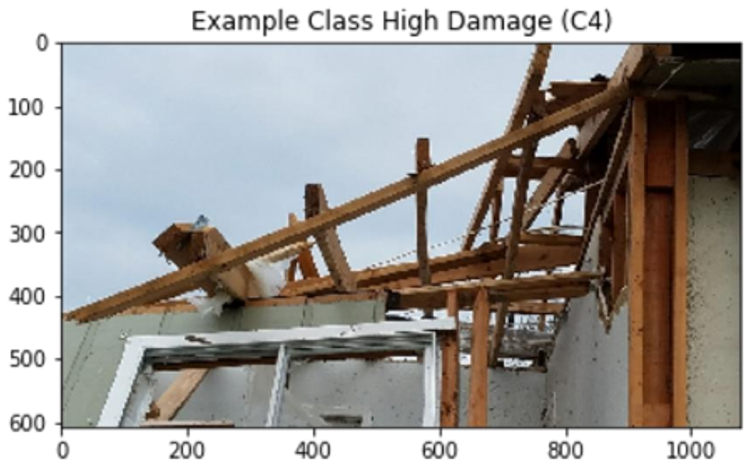
- Go to NaturalHazardPrediction
  - Run copy-data.ipynb
- Go to NaturalHazardPrediction/pytorch/
  - Run torch-train-1st.ipynb

Hands-on Exercise

Image Classification with Hurricane Harvey Dataset







Hands-on Exercise

- What is limiting the model performance?

	2 Categories	3 Categories	5 Categories
Val_acc	92%	72%	42%

- Model capacity
- Data
- Quality
- Imbalance among categories
- Others