Throughput Machine Learning

Ian Ross

Data Engineer, Center for High Throughput Computing

CHTC Center For High Throughput Computing

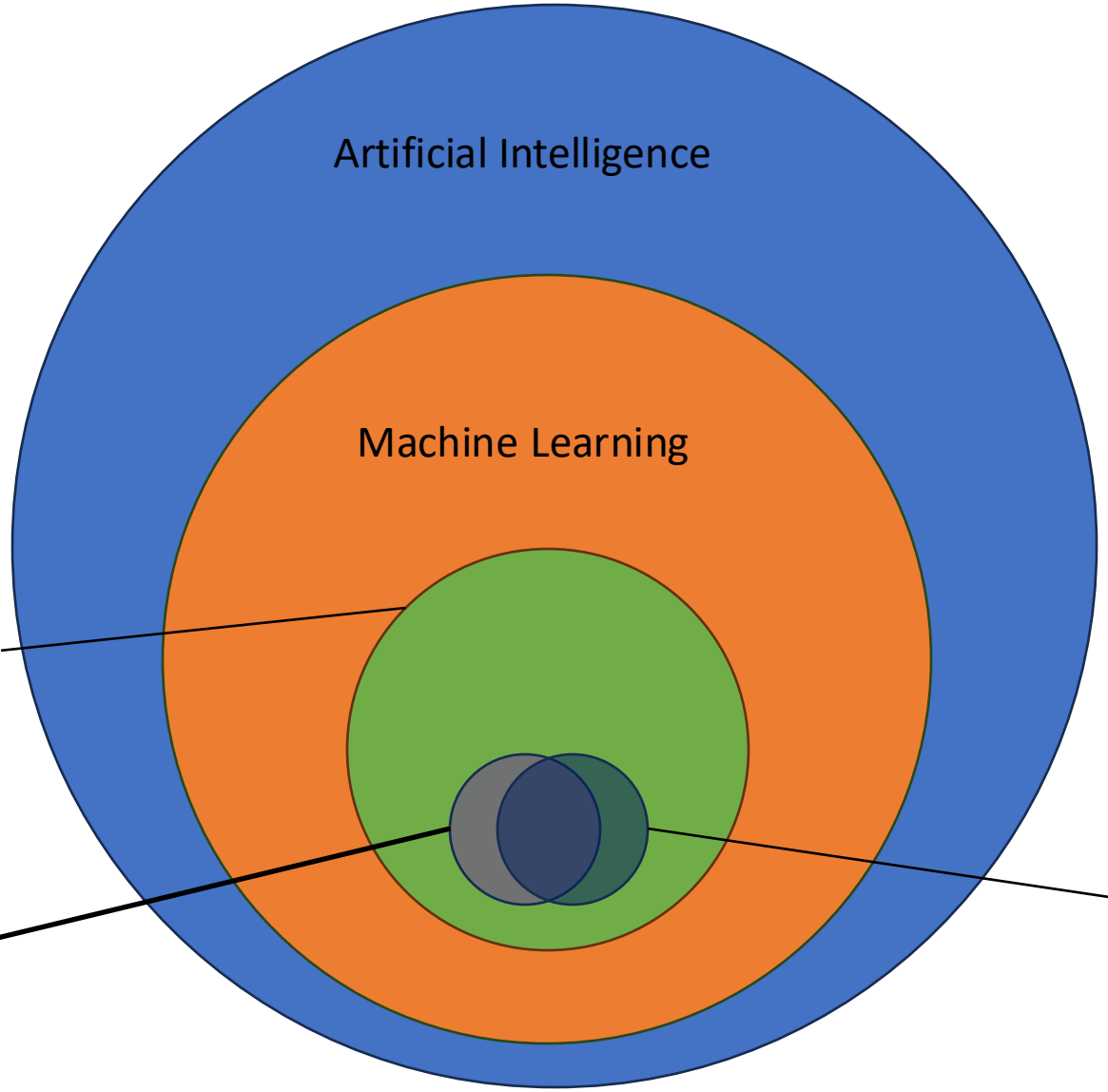
Objectives

- Provide a too-brief overview of artificial intelligence and machine learning (AI, ML)
- Throughput Machine Learning
- Planting some seeds
  - Is my ML work “HTC-able”?
  - If yes, how can I go about growing it?

AI/ML – a too-brief overview

- **Artificial intelligence** – Methods and software to enable machines to *observe, identify, and react to stimuli to achieve a defined goal*
- **Machine learning** – Algorithms and practices to enable machines to recognize patterns in data and generalize to new data to achieve tasks with or without *supervision*
  - Subset of AI

smbc-comics.com



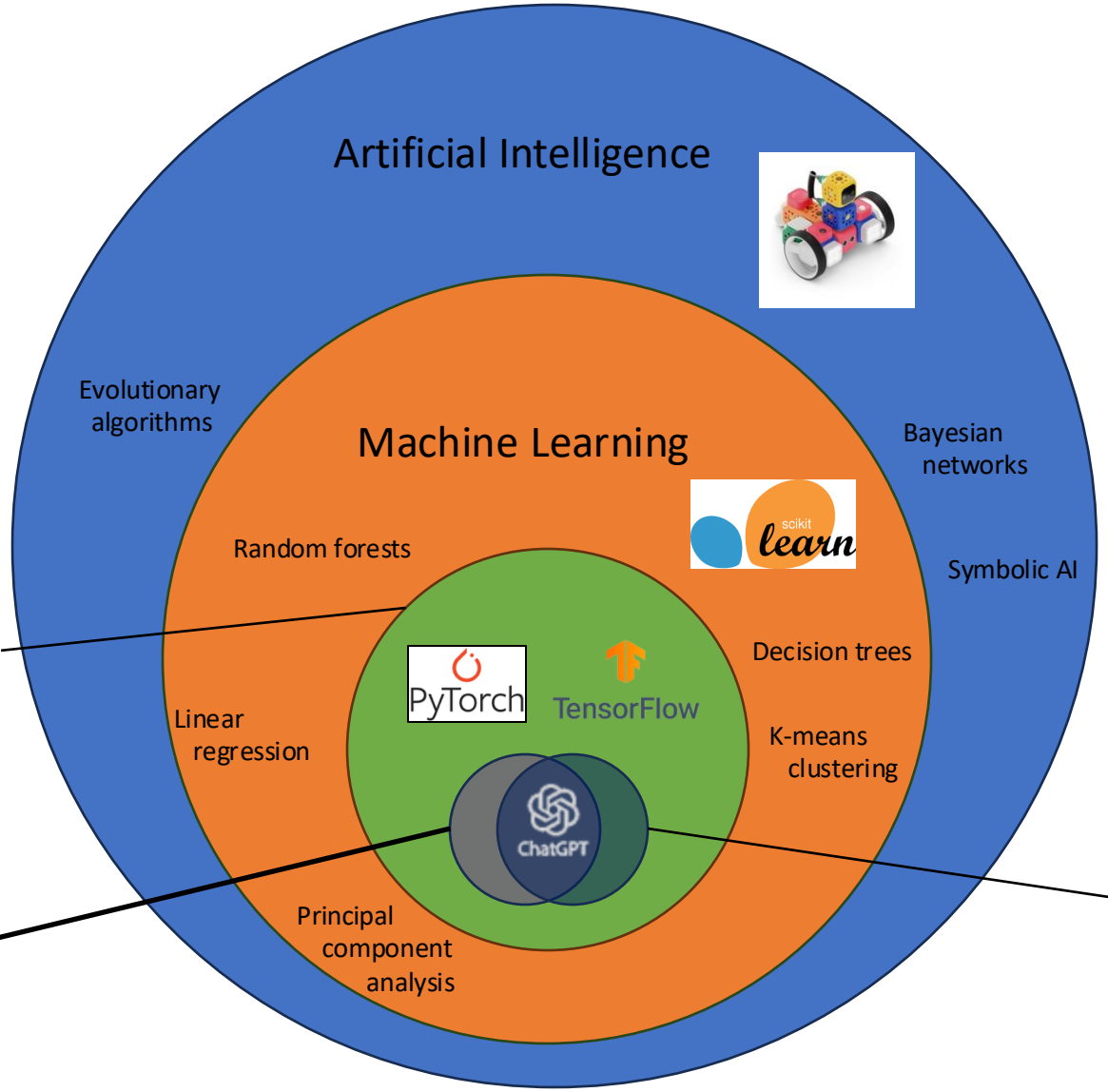
Deep learning

Using **artificial neural networks** to capture connections and strengths (weights) between inputs and outputs

Large language models

Generative AI

Models trained to *generate* data similar to that used to train



Deep learning

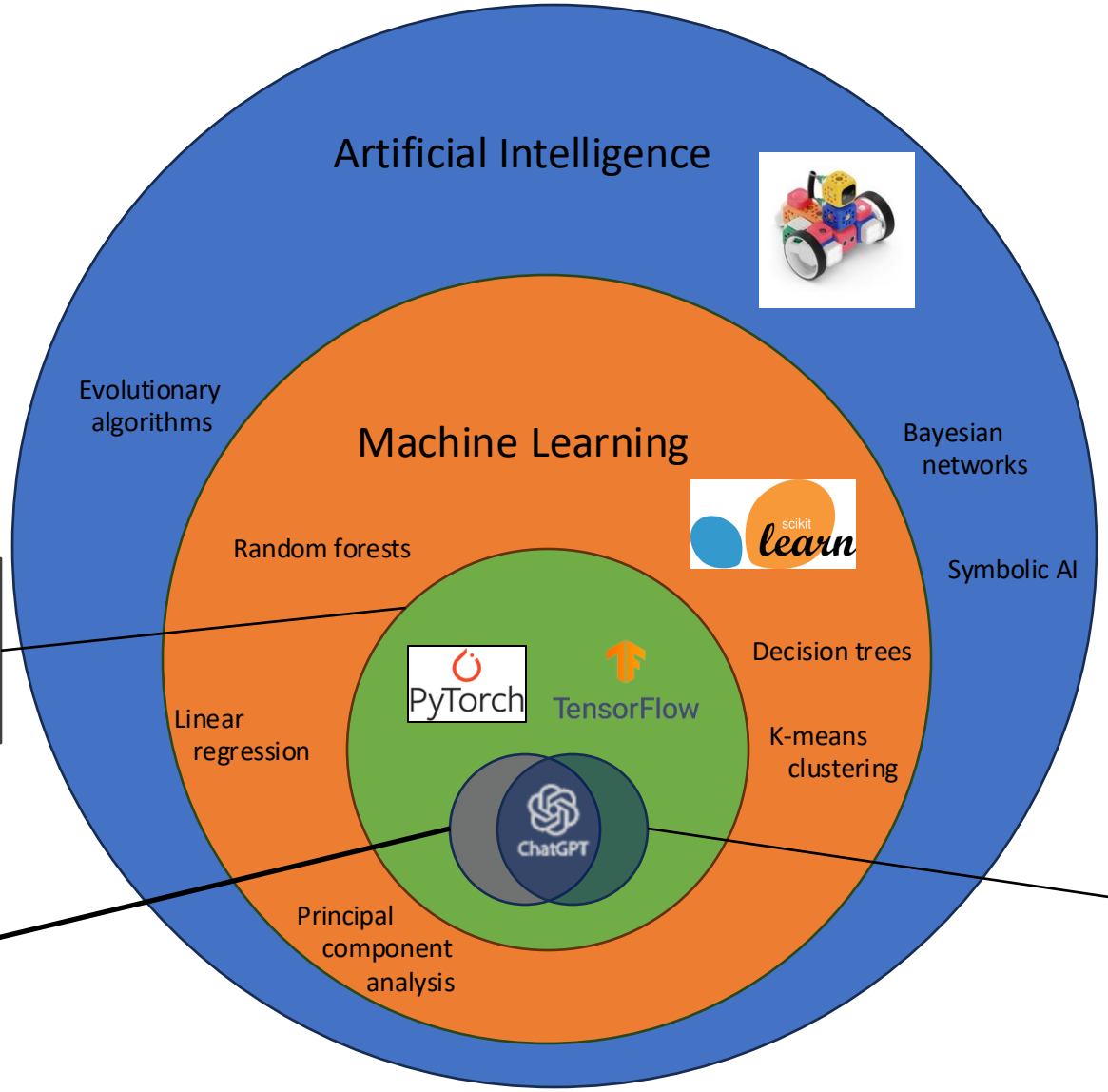
Using **artificial neural networks** to capture connections and strengths (weights) between inputs and outputs

Large language models

Generative AI

Models trained to *generate* data similar to that used to train

Toy robot photo by Robo Wunderkind on Unsplash



Deep learning

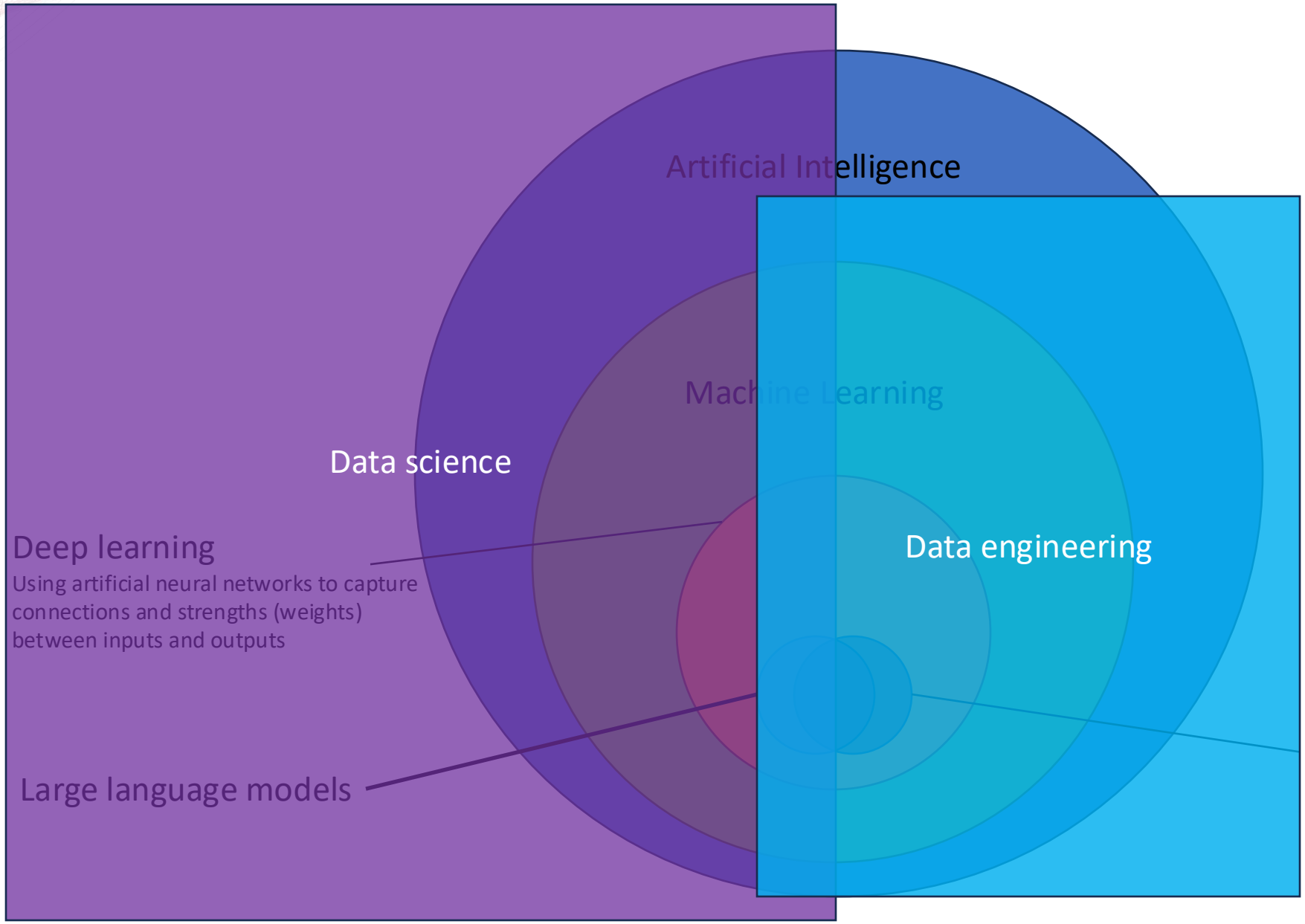
Using **artificial neural networks** to capture connections and strengths (weights) between inputs and outputs

Large language models

Generative AI

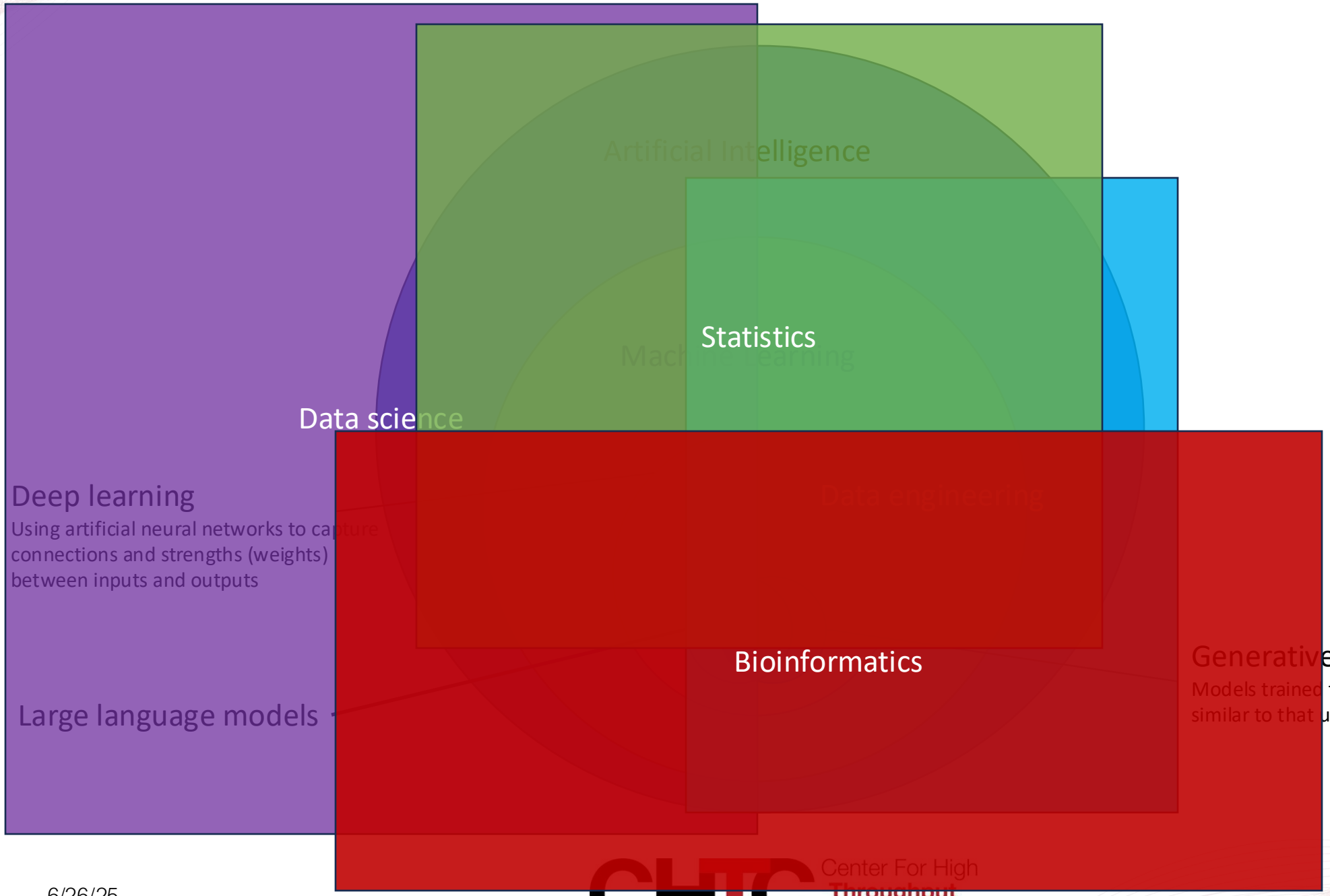
Models trained to *generate* data similar to that used to train

Toy robot photo by Robo Wunderkind on Unsplash



Generative AI

Models trained to *generate* data similar to that used to train

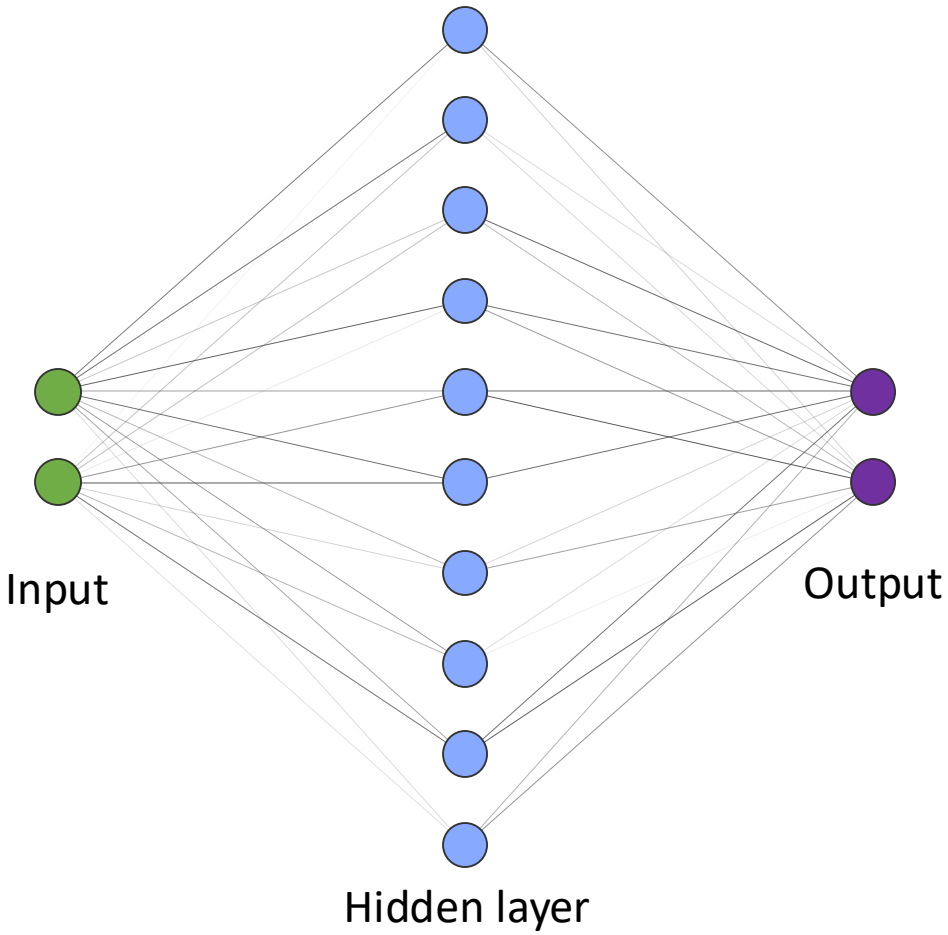

Artificial Intelligence
Statistics
Machine Learning
Data science
Data engineering
Bioinformatics
Deep learning
Using artificial neural networks to capture connections and strengths (weights) between inputs and outputs
Large language models
Generative AI
Models trained to *generate* data similar to that used to train

We mostly care about these disciplines primarily as tools and techniques that enable new SCIENCE

…but there are foundations to understand (and complications to tackle!)

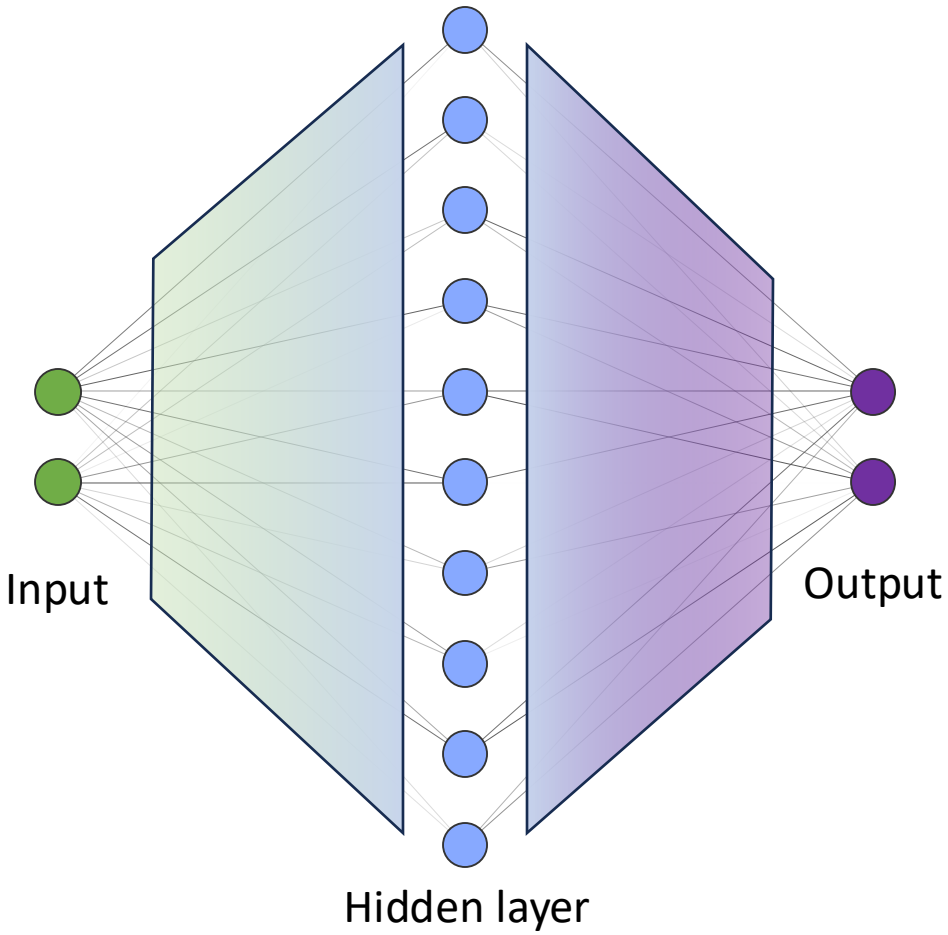
Neural networks, simplified

- Historically analogous to neurons in a brain, where electrical signals spark pathways to carry signals
- Artificial neurons are arranged into *layers*, (input, output, or hidden)
- The *activation* (value) of a neuron depends on the values of the *previous layer* and *predefined weights*



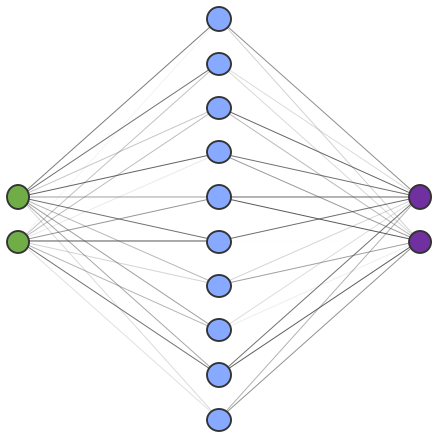
Neural networks, simplified

- These *weights* are tuned during *training*, in order to maximize success for the defined task
- These architectures can get *very* complicated, and this is where a lot of recent innovation is happening
  - Attention mechanisms (transformers), RNN, (R)-CNNs, LSTM, …

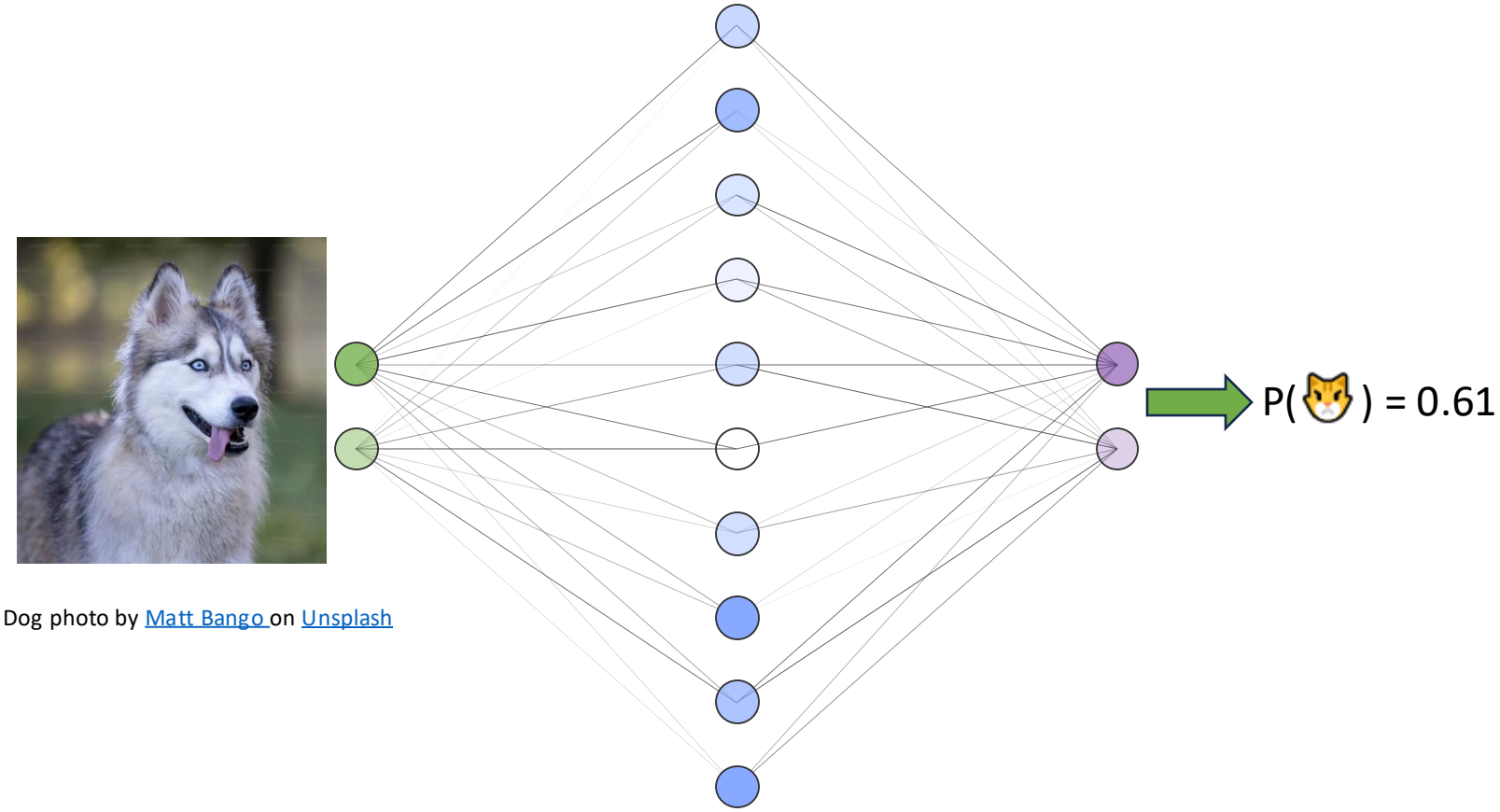


Dog/cat classifier “example”

- Imagine we had thousands of pictures of our pets (and our friends’ pets and our friends’ friends’ pets and…)
- We might initialize a set of weights and then start the training process…



Training, oversimplified



Dog photo by Matt Bango on Unsplash

Push training data into the model, compare the outputs to the truth, calculate how the predictions change as we shift the weight parameters slightly, move the parameters in a promising direction, and repeat. And repeat. And repeat…

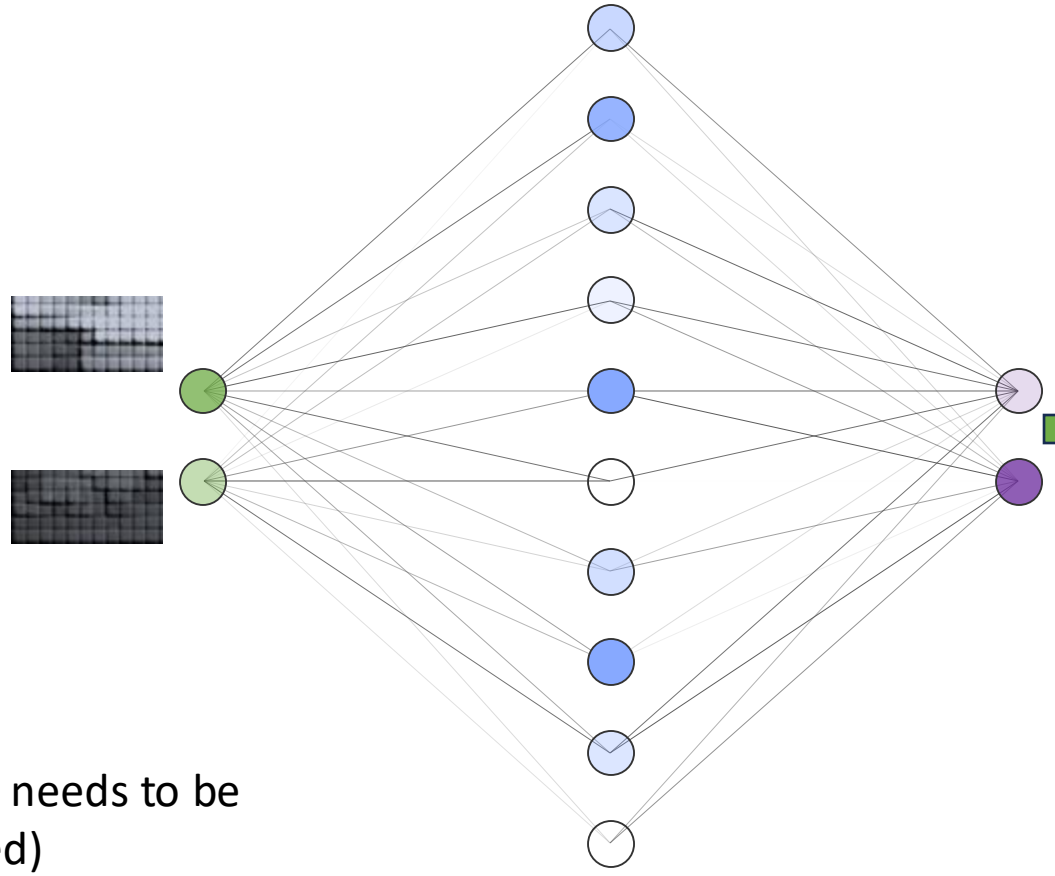
This is computationally expensive, especially as models get larger and more complex (e.g. 30.85 *million* GPU hours for Llama3.1 405B)

Now we have a model!

- Let's deploy it and identify some pets!
- We'll pass in some photos from CHTC's #social channel (***not from the training set***) and see what our model *infers* from these pictures

Best friends photo by Alec Favale on Unsplash

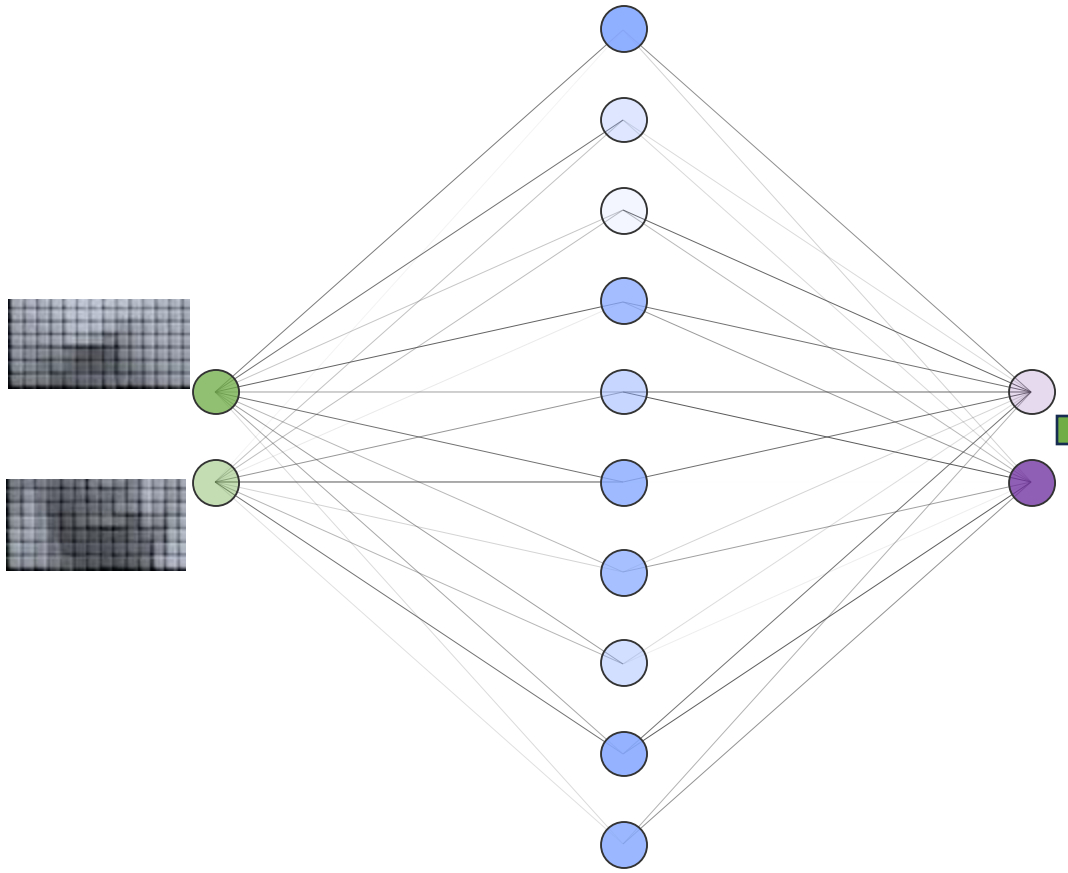
Inference tests



P(🐶) = 0.99

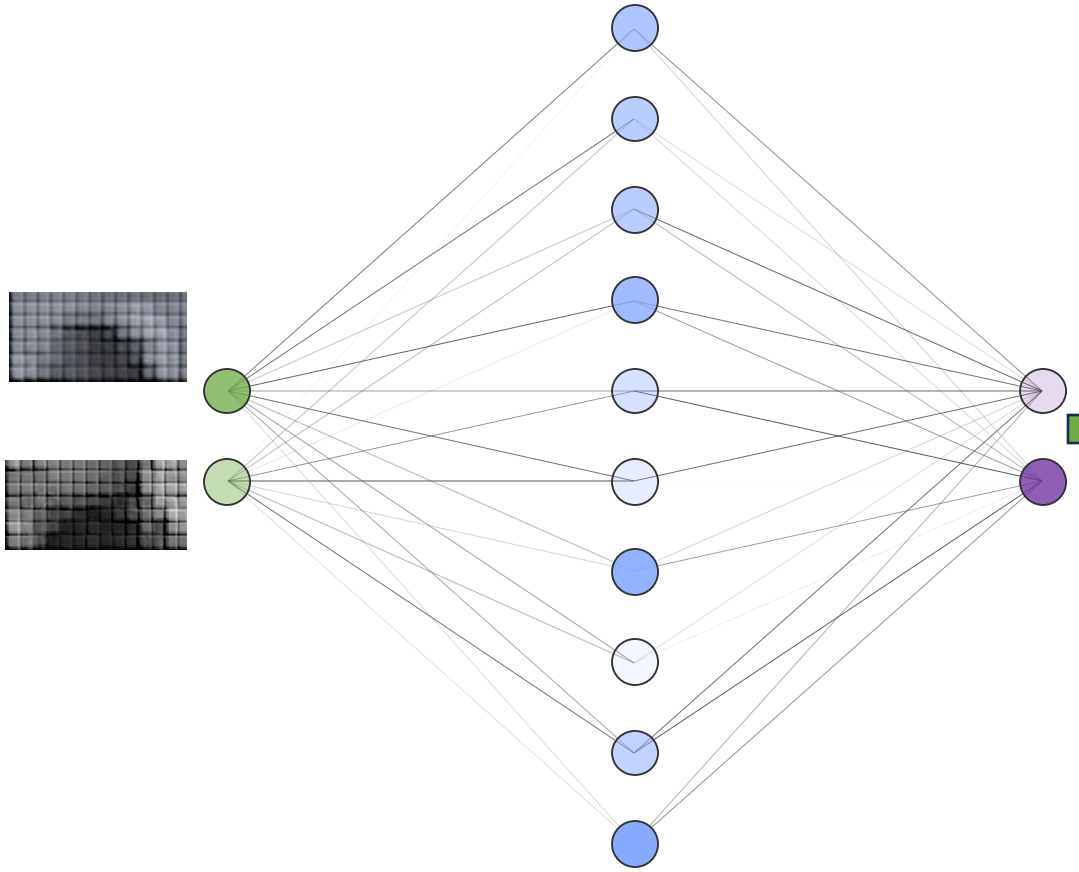
Input data (in training too!) needs to be preprocessed (e.g. tokenized)

Inference tests



P(🐱) = 0.99 👍

Inference tests



P(🐱) = 0.71

Limits of our model

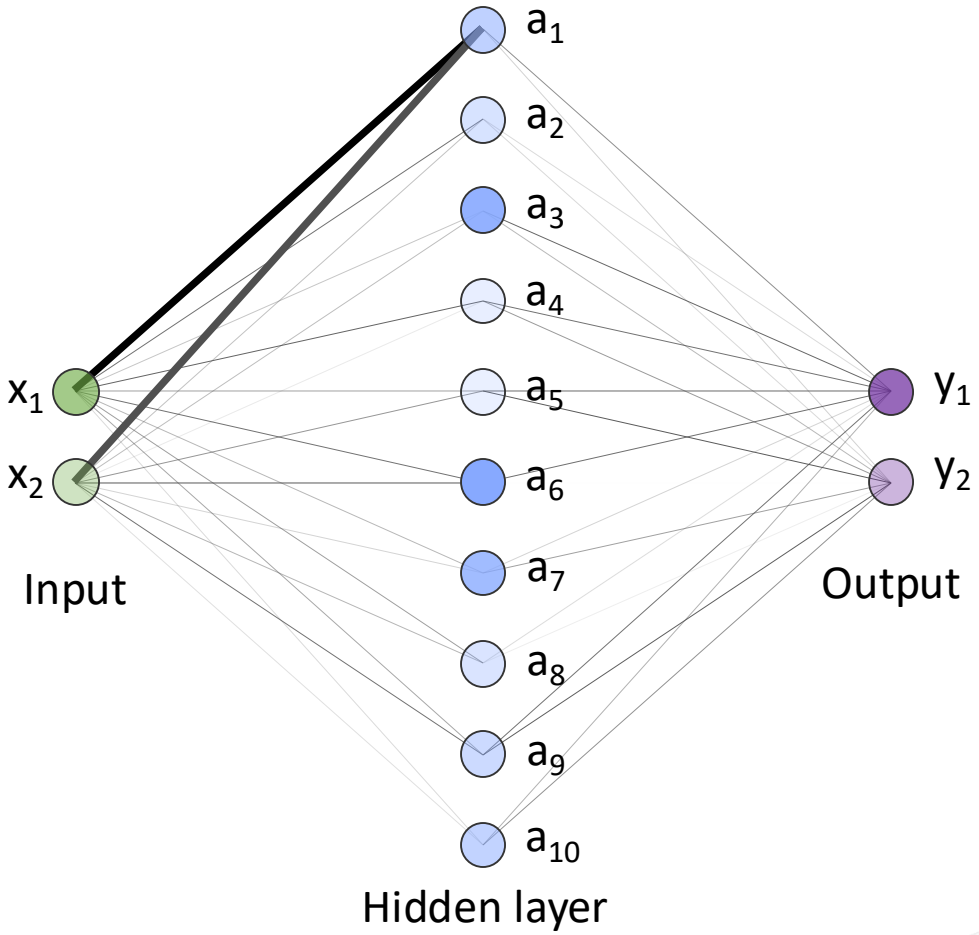
- These weights were trained on our pets dataset. Weights (and therefore the hidden layer) of this model captured *pet essence* .
- Our model *does not know how to identify anything else.* A picture of my son is *outside of the training distribution* and we should not expect it to perform in this case.
- ***Know your data and know your objective!***

- Let's take a closer look at what this inference requires computationally..

A bit of math…

- Nodes are a linear combination of weighted nodes from the previous layer:

$$a_1 = W_{11} * x_1 + W_{21} * x_2$$



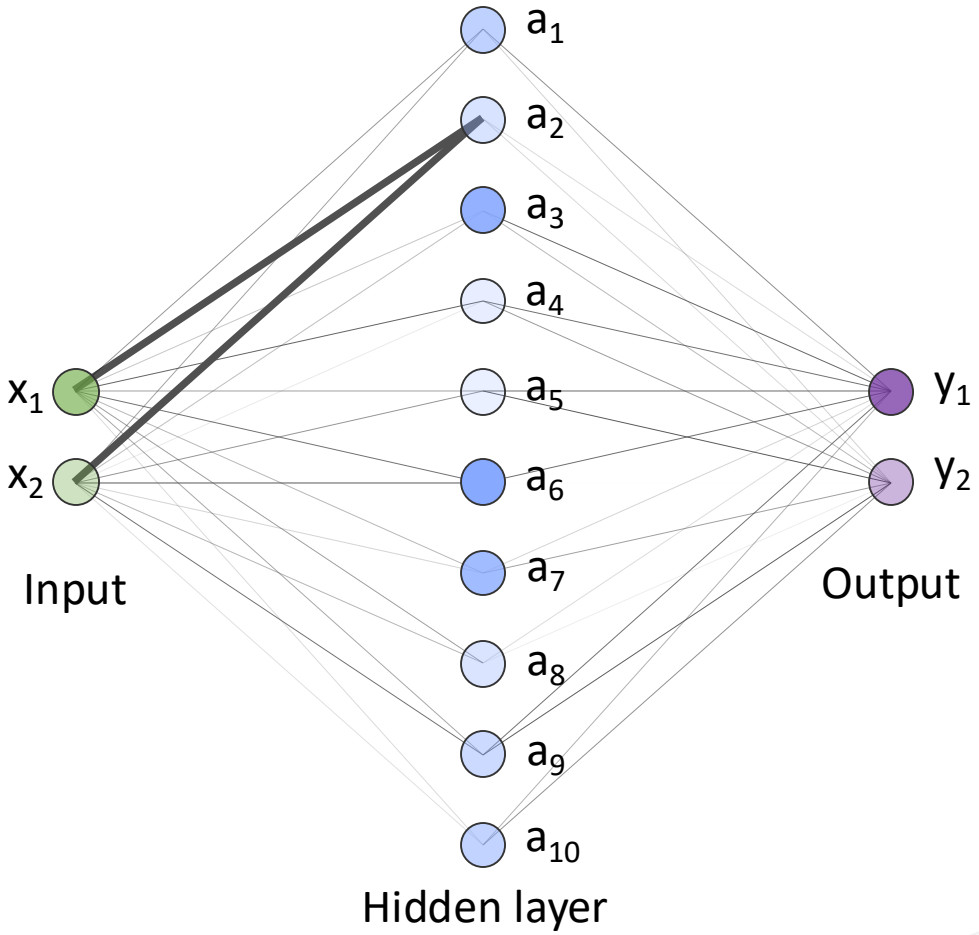
$a_1 = W_{11} * x_1 + W_{21} * x_2$

$a_2 = W_{12} * x_1 + W_{22} * x_2$

…

$a_n = W_{1n} * x_n + W_{2n} * x_n$

…and similar for the y_n



So… why GPUs?

GPUs are successful because they do one thing really well:

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

Inference and training often boil down to ***parallelized matrix multiplication***, which GPUs excel at.

Challenges in ML/AI workflows

These are not **new** challenges! Defining resource needs, scheduling, data movement, workflow orchestration, learning new technologies… Sound familiar? So let's talk through examples to understand whether an ML task is a good candidate for HTC…

Throughput machine learning – Use case 1

I have 18 million scientific articles, and I want to search for, extract, and synthesize visual artifacts across them!

Is this a throughput workflow?

What is the smallest, self-contained computational task that is part of your work? How big is the list of these tasks? What is needed to run one task?

Scaling out with HTC

The atomic task is running our COSMOS visual pipeline on an individual PDF. Each PDF is on the order of 1 MB, produces 4 MB of output, and takes 5 seconds on a GPU or 5 minutes on a CPU. The model is ~8GB, and I have a docker container ready to go. However, these PDFs are bound by publisher agreements and can't leave UW campus.

Great! This sounds ideal for CHTC, with standard input file transfer mechanisms. Have fun!

Section

Body Text(0.9924252033233643)

One script sets up the initial condition and runs the model:

```
./runHalfar.py
```

Note that to run the test with the `halfar-H0.config` settings, you can use the `-c` commandline option for specifying a configuration file:

```
./runHalfar.py -c halfar-H0.config
```

Another script analyzes and plots the results:

```
./halfar_results.py
```

Section

Body Text(0.9994284510612488)

With the default .config settings, this simulation should only take a few seconds and is a good first test for a working Glide dycore. With Glissade, the Blatter-Pattyn option takes a few minutes, but the SIA and L1L2 settings are much faster. As the dome of ice evolves, its margin advances and its thickness decreases (there is no surface mass balance to add new mass). The script `halfar_results.py` will plot the modeled and analytic thickness at a specified time (Figure 8.1), and also report error statistics. Invoke `halfar_results.py --help` for details on its use.

Figure(0.9414491057395935)

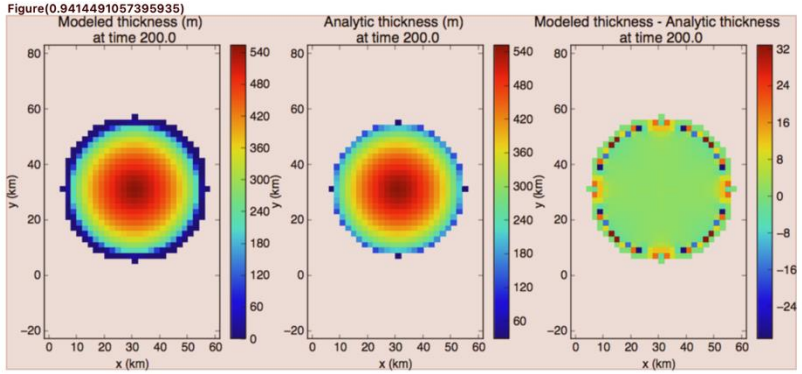


Figure Caption(0.5313378572463989)

Figure 8.1: Halfar test case results (using Glide) after 200 years of dome evolution. This figure is generated by `halfar_results.py`.

Section

Body Text(0.9699838757514954)

This test case is from phase 1 of the European Ice Sheet Modelling INiTiative intercomparison experiments. These experiments are described in more detail here[2] and in Huybrechts et al. (1996)

Throughput machine learning – Use case 2

I want to train many models, empirically measure their predictive power, and use those models to drive scientific exploration.

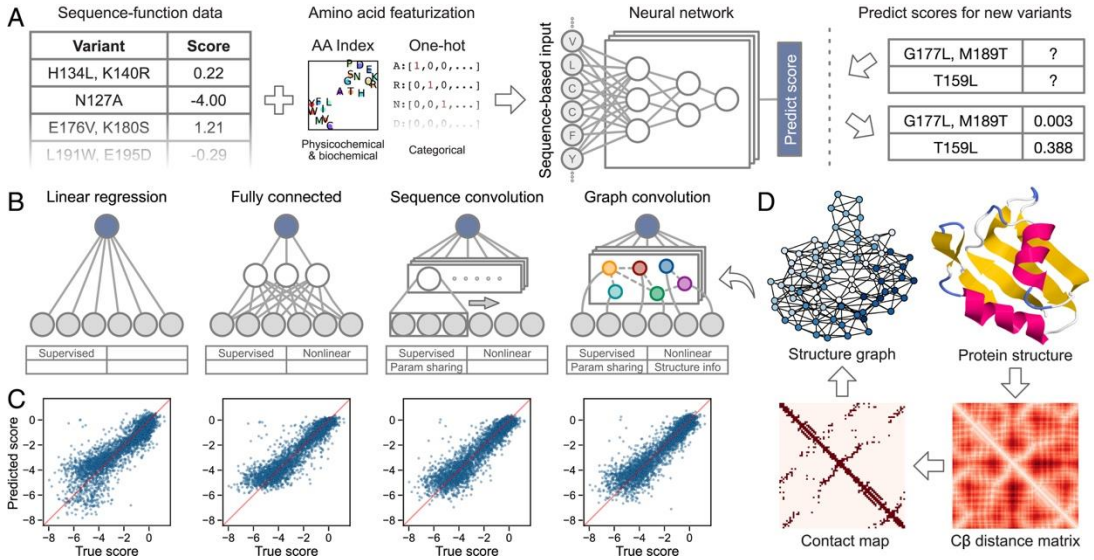
Is this a throughput workflow?

What is the smallest, self-contained computational task that is part of your work? How big is the list of these tasks? What is needed to run one task?

Scaling out with HTC

The atomic task is training a single model from our dataset, which is 10GB. We want to test many different architectures, but anticipate GPU runtimes on the order of days for each model. We want to test as many model architectures as possible. CPU, memory, and disk requirements are minimal.

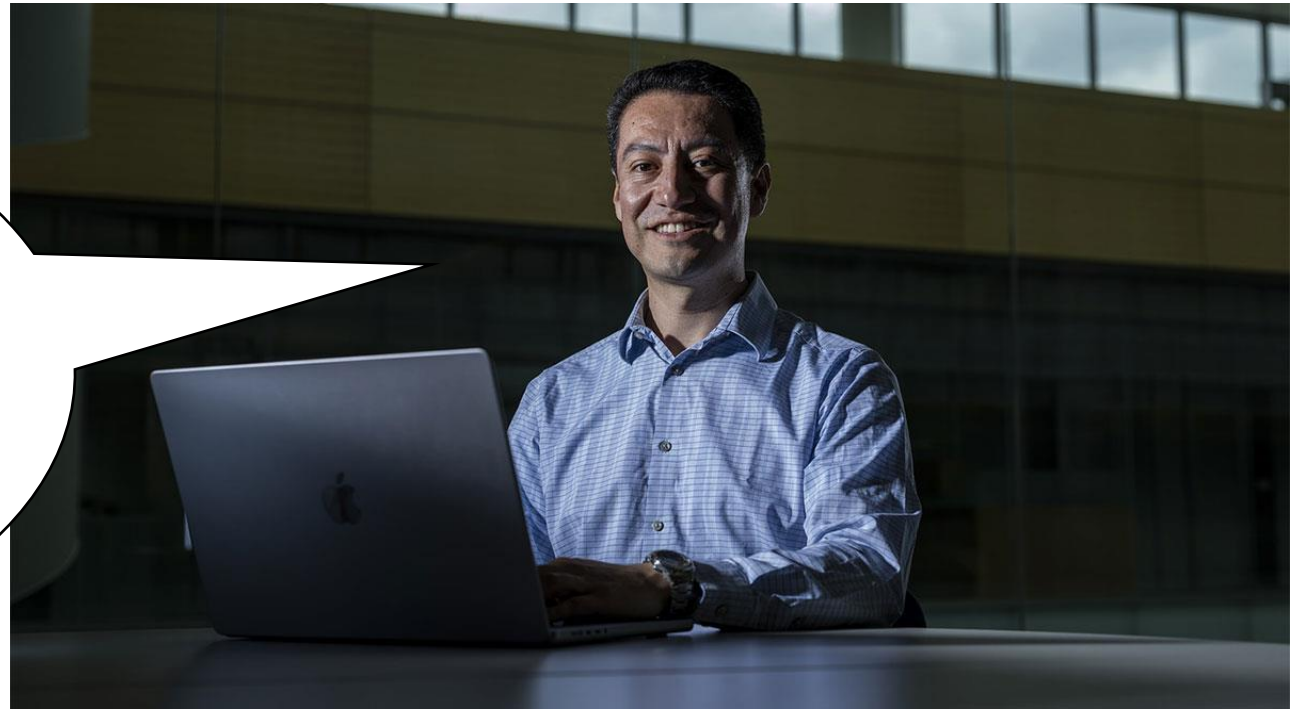
Welcome to the OSPool! Let's learn about OSDF and job checkpointing!



https://www.pnas.org/doi/full/10.1073/pnas.2104878118

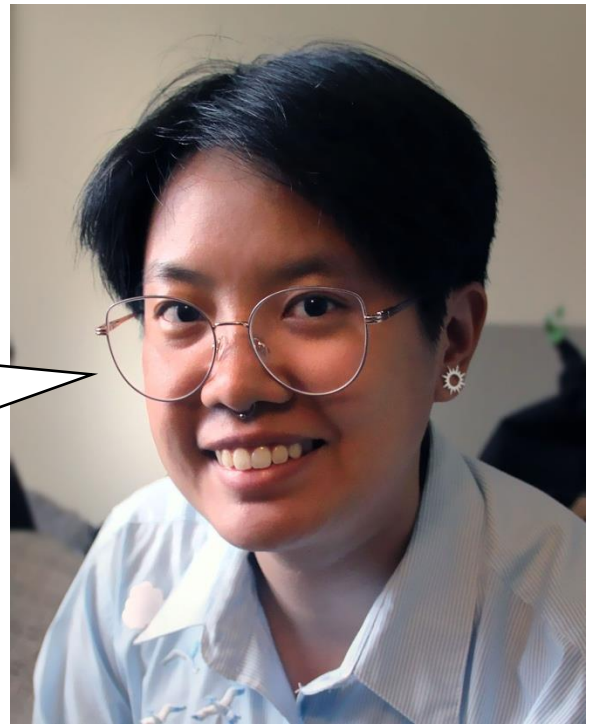
Throughput machine learning – Use case 3

I want to create a foundation model for bioimaging and want to scale training across multiple nodes!



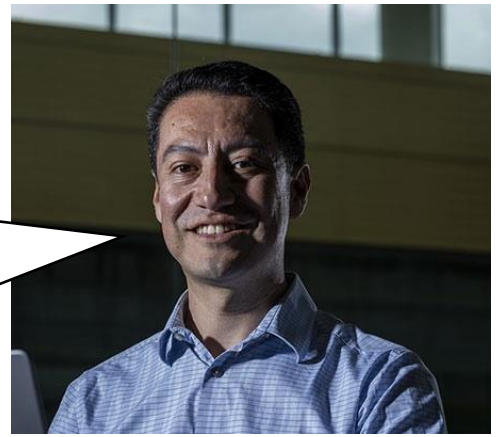
Is this a throughput workflow?

What is the smallest, self-contained computational task that is part of your work? How big is the list of these tasks? What is needed to run one task?



Scaling out with HTC?

Our dataset is 2TB. The model architecture we want to use is too big to fit on one GPU, and the memory needs are on the order of 128GB, and an epoch of training takes days



This isn't a great fit for our usual computing philosophy, but let's talk more and work together to see what we can do!

Throughput machine learning – use case 4

I want to actually create our theoretical cat-dog classifier example! I've talked to my RCF friends and determined it's a good fit for CHTC! Let's go!

As I roll up my sleeves and get to work, what do I need to consider…

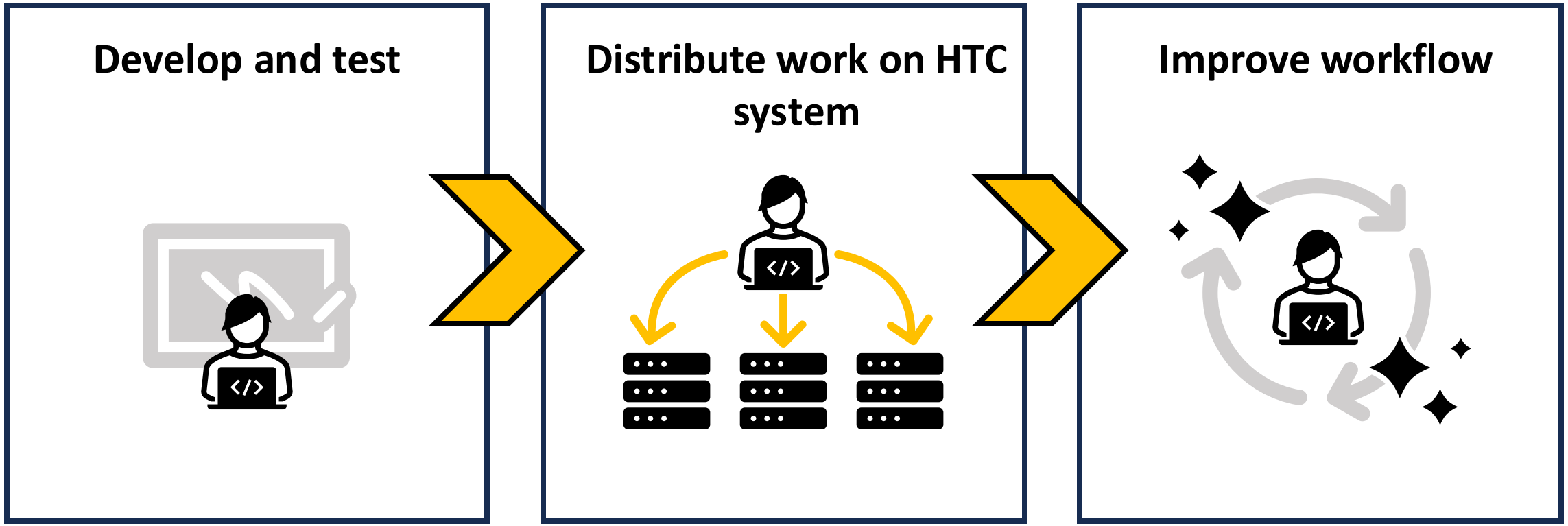
…as I start to do development?

…as I think about distribute jobs in CHTC?

…as I think about improving my workflow in CHTC?

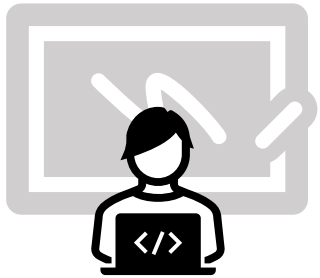
This example and walkthrough is available at https://github.com/CHTC/templates-GPUs/tree/master/ml_workflow

Researcher-to-AI-workflow-proficiency pipeline!



What should you consider during development?

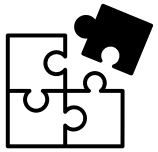
Develop and test



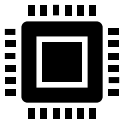
Objective

Get a *minimally viable workflow* running on a local machine, while laying a *foundation for distributed work* that will ***accomplish your science.***

What should you consider during development?



Test with a subset of data.



Consider time, space, memory, and GPU needs.

Reduce computational project to a minimally viable workflow.

Know your software and create the software environment.

The training script

Define the model architecture

```
class CatAndDogConvNet(nn.Module):
    def __init__(self):
        super().__init__()

        # convolutional layers (3,16,32)
        self.conv1 = nn.Conv2d(in_channels = 3, out_channels = 16,
kernel_size=(5, 5), stride=2, padding=1)
        self.conv2 = nn.Conv2d(in_channels = 16, out_channels = 32,
kernel_size=(5, 5), stride=2, padding=1)
        self.conv3 = nn.Conv2d(in_channels = 32, out_channels = 64,
kernel_size=(3, 3), padding=1)

        # connected layers
        self.fc1 = nn.Linear(in_features= 64 * 6 * 6,
out_features=500)
        self.fc2 = nn.Linear(in_features=500, out_features=50)
        self.fc3 = nn.Linear(in_features=50, out_features=2)
    def forward(self, X):
        …
```

```
class CatDogDataset(Dataset):
    def __init__(self, image_paths, transform):
        super().__init__()
        self.paths = image_paths
        self.len = len(self.paths)
        self.transform = transform
    …
    def __getitem__(self, index):
        path = self.paths[index]
        image = Image.open(path).convert('RGB')
        image = self.transform(image)
        label = 0 if 'cat' in path else 1
        return (image, label)
```

Define dataset handler

Add command line arguments to handle input/output directories

```
def main(
    data_dir: Path = typer.Option("./data", "--data-dir", "-d", \
                help="Directory containing the training data"),
    checkpoint_dir: Path = typer.Option("./checkpoints", "--checkpoint-dir", "-c", \
                help="Directory to save model checkpoints"),
    epochs: int = typer.Option(10, "--epochs", "-e", \
                help="Number of training epochs"),
):
…
    device = torch.device("cuda" if torch.cuda.is_available() else "mps" if torch.backends.mps.is_available()
else "cpu")
```

Use cuda as the backend device, if possible.
(⭐ bonus: use MPS if running on his Macbook! ⭐)

```
    start = time.time()
```

Train the model and save it after all epochs!

```
    # Model training...
    start = time.time()  # start timer for training loop
```

Define dataset, loader, initialize model, and move to the specified device

```
            X, y = X.to(device), y.to(device)

            preds = model(X)
            loss = loss_fn(preds, y)

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

            accuracy = ((preds.argmax(dim=1) == y).float().mean())
            epoch_accuracy += accuracy
            epoch_loss += loss
            print('.', end='', flush=True)
    # save model
    torch.save(model.state_dict(), model_path)
```

Defining the environment and container image

Dockerfile

```
# Start from a standard CUDA image
FROM nvidia/cuda:12.4.1-runtime-ubuntu22.04

# Update software repositories and install dependencies
RUN apt-get update
RUN apt-get install software-properties-common -y
RUN add-apt-repository ppa:deadsnakes/ppa -y
ARG DEBIAN_FRONTEND="noninteractive"
ENV TZ=America/Chicago
RUN apt-get update

# Install python and pip
RUN apt-get install -y \
    python3.12 python3.12-dev python3.12-venv zip \
    && update-alternatives --install /usr/bin/python python
/usr/bin/python3.12 1 \
    && python -m ensurepip --upgrade \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy only the requirements file first and install dependencies
COPY requirements.txt .
RUN pip3.12 install -r requirements.txt
```

- We want to use GPUs and this is a generally compatible CUDA image to start from
- We want to install python3.12 and make sure that pip is available
- Install our python dependencies

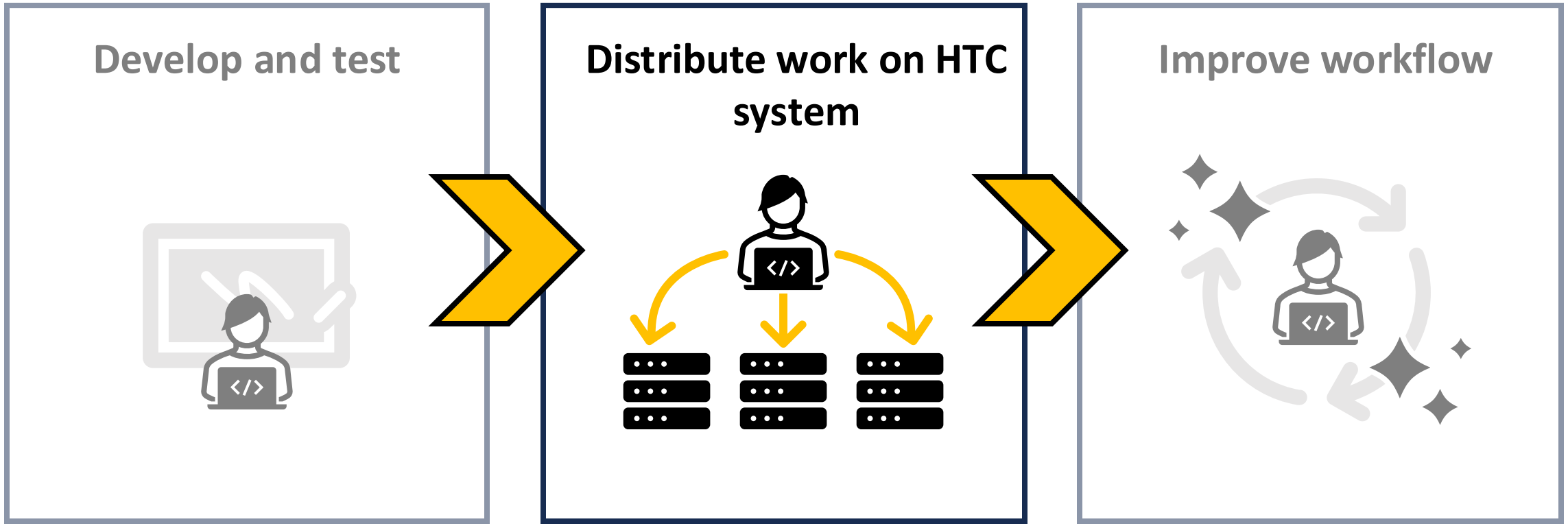
requirements.txt

```
pandas
typer
torch
matplotlib
torchvision
```

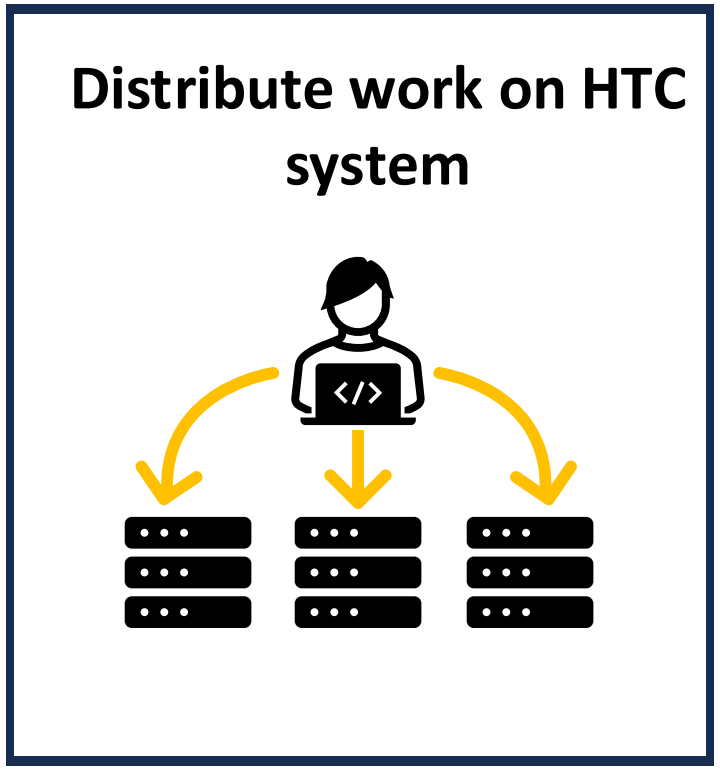
Some development time passes…

- My code and my environment are ready!
- I've gotten deeper understanding into
  - My data
    - I've split it into training and validation sets, defined preprocessing, and staged It appropriately
  - My resource needs
    - Or I've run enough tests for a useful estimate
- And I've done some reps training locally
  - At least enough to know that the code works

Researcher-to-AI-workflow-proficiency pipeline!



What should you consider during deployment?



Objective

Effectively utilize the available **resources** to train the model (or many variants of the model).

GPUs in the OSPool

The OSPool makes a variety of GPUs available to you (just don't ask how many!)

Some GPUs you might land on:

- GeForce GTX 1080 Ti (Capability: 6.1)
- V100 (Capability: 7.0)
- GeForce GTX 2080 Ti (Capability: 7.5)
- Quadro RTX 6000 (Capability: 7.5)
- A100 (Capability: 8.0)
- A40 (Capability: 8.6)
- GeForce RTX 3090 (Capability: 8.6)

Compute Capability defines specific hardware features on the GPU.

It doesn't tell you about available memory.

GPUs in the GeForce series are "gaming" GPUs, but don't mistake this to mean they're incapable!

Submit file options

- Request GPUs with “request_gpus” and require minimum memory and CUDA capability:

```
request_cpus = 1
request_memory = 4 GB
request_disk = 8 GB
request_gpus = 1

gpus_minimum_capability = 8.0
gpus_minimum_memory = 4000
```

https://portal.osg-htc.org/documentation/htc_workloads/specific_resource/gpu-jobs/

The submit file

train.sub

```
container_image = osdf:///ospool/ap40/data/iaross/catdog.sif
# see our guide on converting docker images to apptainer

request_disk = 500MB
request_memory = 6GB
request_cpus = 1

# See our CHTC guide on GPU usage / OSPool guide
request_gpus = 1
gpus_minimum_capability = 7.5
gpus_minimum_memory = 4096

executable = train.sh

transfer_input_files = train.py,
         osdf:///ospool/ap40/data/iaross/cat_dog/train.zip
transfer_output_files = output

output = $(CLUSTERID).out
error = $(CLUSTERID).err
log = catdog_training.log

queue
```

train.sh

```
#!/bin/bash

wd=$(pwd)
mkdir data
mkdir $wd/output

unzip train.zip -d data/
rm train.zip

cd /app/

python train.py \
  --data-dir $wd/data/ \
  --checkpoint-dir $wd/output/
```

GPUs in CHTC CHTC

There are ~300 GPUs in CHTC, ranging from GTX 1080Ti to H200

Opt into shared GPUs

- +WantGPULab = true
- +GPUJobLength = "short" (or "medium" or "long")

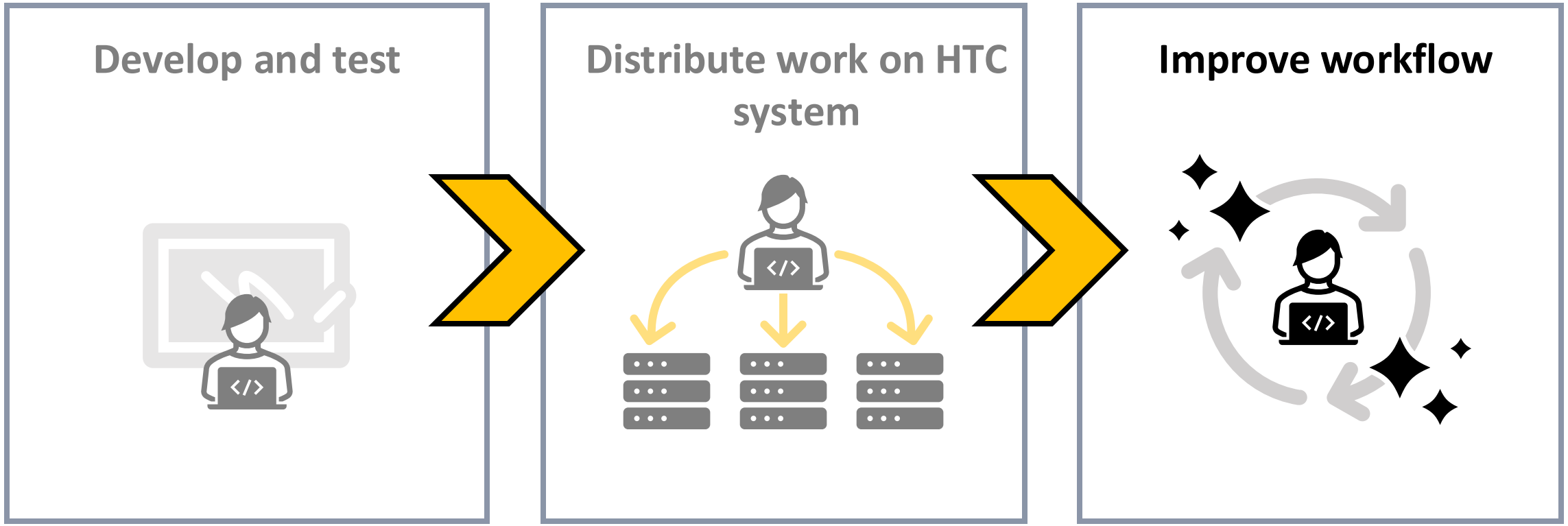
Backfill researcher-owned GPUs

- Approximately half are owned by specific groups, but you can opt into backfilling them
- +IsResumable = true

GPUJobLength	Maximum runtime	Per-user limitation
Short	12 hours	2/3 of GPUs
Medium	24 hours	1/3 of GPUs
Long	7 days	4 GPUs

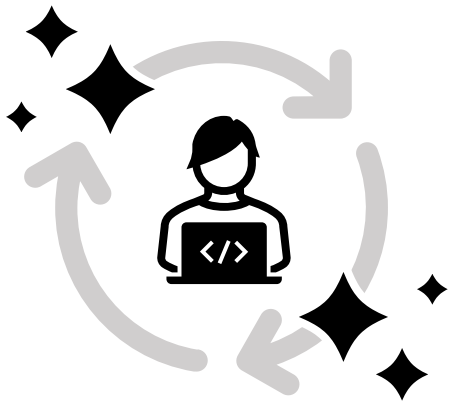
*Maximize your workload capacity with **shorter** jobs!*

Researcher-to-AI-workflow-proficiency pipeline!



How can you improve your workflow?

Improve workflow

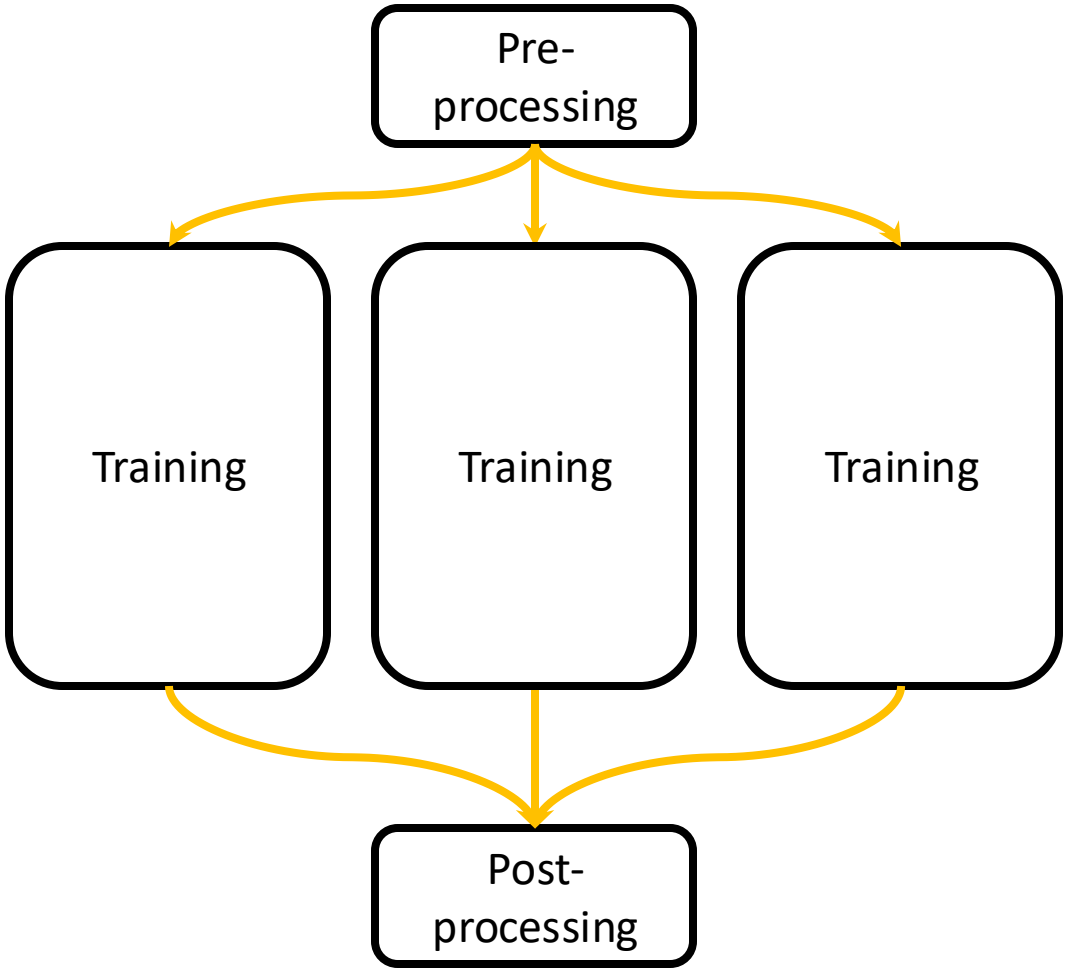


Objective

Get even more out of the available resources ("do more with less"), monitor and organize runs, and/or create ensembles of models.

How can you improve your workflow?

- Automated workflows (e.g. using DAGMan)
- Hyperparameter and ensembles – Don't think of training one model, think of training many and finding (or combining) the best
- Weights and Biases (or similar tools) to monitor training runs
  - Works as expected, but be aware of API key leakage



Checkpointing

- Update the logic of the training script to enable loading+resuming from a checkpoint.
- The job doesn't need to run to full completion within one job cycle. This means:
  - Resilience against job eviction and machine issues
  - Ability to request "short" jobs (more slots available!)

Where to go from here?

- Think about your workflow!
- A world of education opportunities
  - Pytorch, HuggingFace, <your software of choice> documentation
  - YouTube for explanations and theory
  - Blogs for examples and inspiration
  - arXiv for preprints
- Explore available pre-made images: Docker Hub, NGC catalog
- OSPool documentation
- CHTC documentation

Questions?

- Talk to us! Don’t let computing be a barrier to your research!