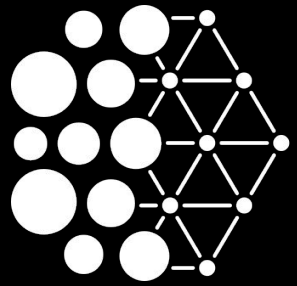
Quebec
Artificial
Intelligence
Institute

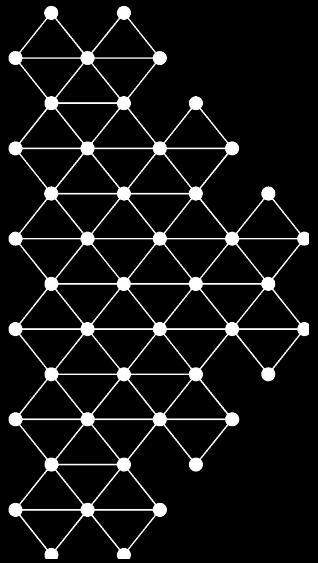


Mila

Machine Learning Tools - Updates
Week 1 - Part 2

Jeremy Pinto
jeremy.pinto@mila.quebec

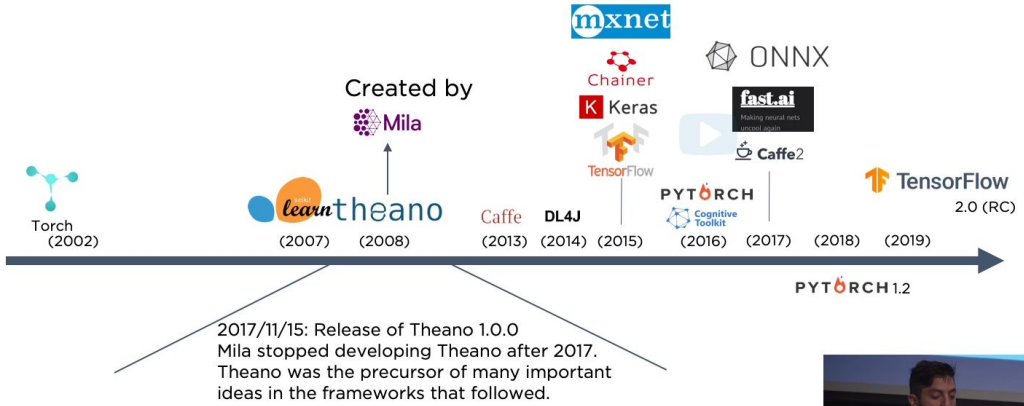
Quebec
Artificial
Intelligence
Institute



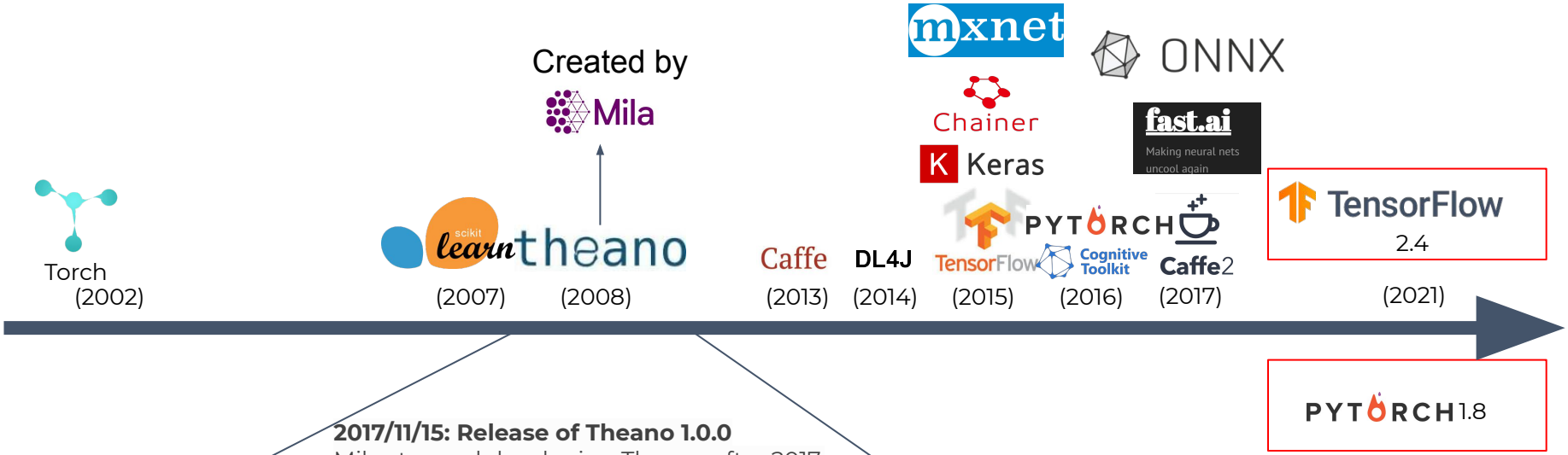
Frameworks - Where are we now?

Frameworks - where are we now?

History of Deep Learning Frameworks



Frameworks - where are we now?




SCORPION
67
SHANG TSUNG
4 HITS
22% Damage
+4
VS

Pytorch vs. tensorflow

- It’s becoming increasingly hard to say which is “better”
- There used to be clear advantages/disadvantages
- There are very few differentiators left between both.

What matters most:

- What are the key requirements of your project?
- Has model X already been implemented in pytorch/tensorflow? How good is the implementation? Can you modify it easily?

Pytorch vs. tensorflow

Model declaration: can you spot the differences?

```
# Model Definition
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2d(in_channels=1, out_channels=32,
kernel_size=3)
        self.flatten = Flatten()
        self.d1 = Linear(21632, 128)
        self.d2 = Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.flatten(x)
        x = F.relu(self.d1(x))
        x = self.d2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
# Model Definition
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(filters=32, kernel_size=3,
activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        output = self.d2(x)
        return output
```

```
# Instantiate Model, Optimizer, Loss
model = MyModel()
optimizer = Adam(model.parameters())
loss_object = CrossEntropyLoss(reduction='sum')
```

```
# Instantiate Model, Optimizer, Loss
model = MyModel()
optimizer = Adam()
loss_object = SparseCategoricalCrossentropy(from_logits=True,
reduction='sum')
```

https://towardsdatascience.com/pytorch-vs-tensorflow-in-2020-fe237862fae1

Pytorch vs. tensorflow

Model declaration: can you spot the differences?

```
# Model Definition
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2d(in_channels=1, out_channels=32,
kernel_size=3)
        self.flatten = Flatten()
        self.d1 = Linear(21632, 128)
        self.d2 = Linear(128, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = self.flatten(x)
        x = F.relu(self.d1(x))
        x = self.d2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

```
# Instantiate Model, Optimizer, Loss
model = MyModel()
optimizer = Adam(model.parameters())
loss_object = CrossEntropyLoss(reduction='sum')
```

```
# Model Definition
class MyModel(Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(filters=32, kernel_size=3,
activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10)

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        output = self.d2(x)
        return output
```

```
# Instantiate Model, Optimizer, Loss
model = MyModel()
optimizer = Adam()
loss_object = SparseCategoricalCrossentropy(from_logits=True,
reduction='sum')
```

https://towardsdatascience.com/pytorch-vs-tensorflow-in-2020-fe237862fae1

Libraries

Libraries

torchaudio

torchtext

torchvision

TorchElastic

TorchServe

PyTorch on XLA Devices

PyTorch geometric

Graph Nets library

Libraries and extensions

Explore additional resources to build advanced models or methods using TensorFlow, and access domain-specific application packages that extend TensorFlow.

TensorBoard

A suite of visualization tools to understand, debug, and optimize TensorFlow programs.

TensorFlow Hub

A library for the publication, discovery, and consumption of reusable parts of machine learning models.

Model Optimization

The TensorFlow Model Optimization Toolkit is a suite of tools for optimizing ML models for deployment and execution.

TensorFlow Federated

A framework for machine learning and other computations on decentralized data.

Neural Structured Learning

A learning paradigm to train neural networks by leveraging structured signals in addition to feature inputs.

TensorFlow Graphics

A library of computer graphics functionalities ranging from cameras, lights, and materials to renderers.

Datasets

A collection of datasets ready to use with TensorFlow.

Serving

A TFX serving system for ML models, designed for high-performance in production environments.

Probability

TensorFlow Probability is a library for probabilistic reasoning and statistical analysis.

MLIR

MLIR unifies the infrastructure for high-performance ML models in TensorFlow.

XLA

A domain-specific compiler for linear algebra that accelerates TensorFlow models with potentially no source code changes.

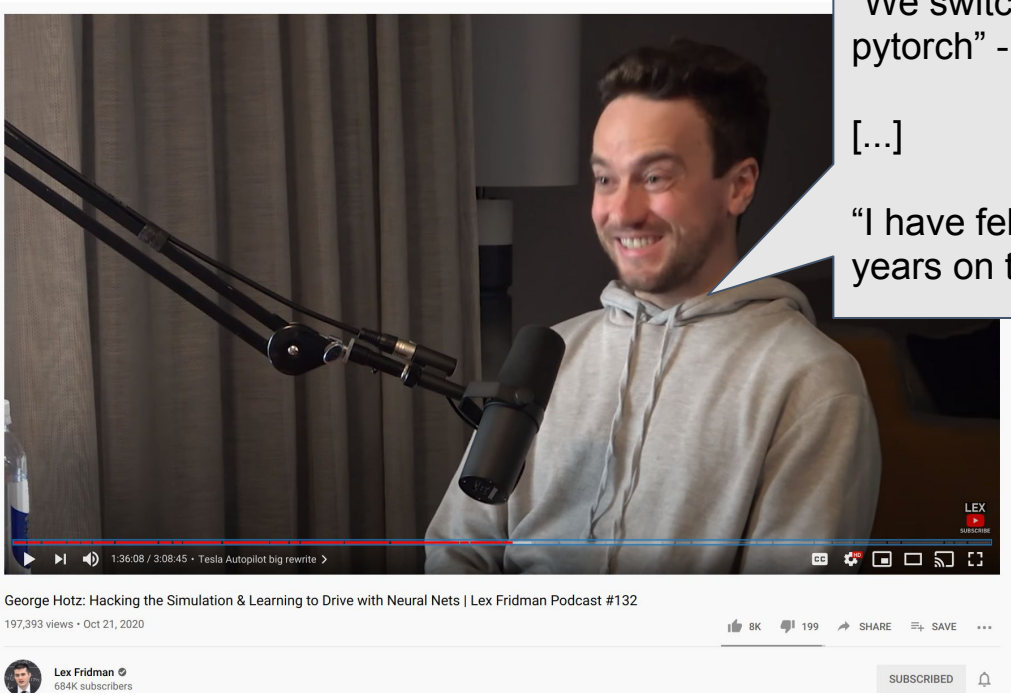
SIG Addons

Extra functionality for TensorFlow, maintained by SIG Addons.

SIG IO

Dataset, streaming, and file system extensions, maintained by SIG IO.

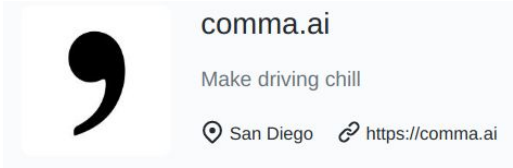
Pytorch vs. tensorflow



"We switched from tensorflow to pytorch" - 1:36:08

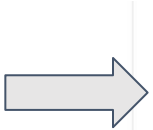
[...]

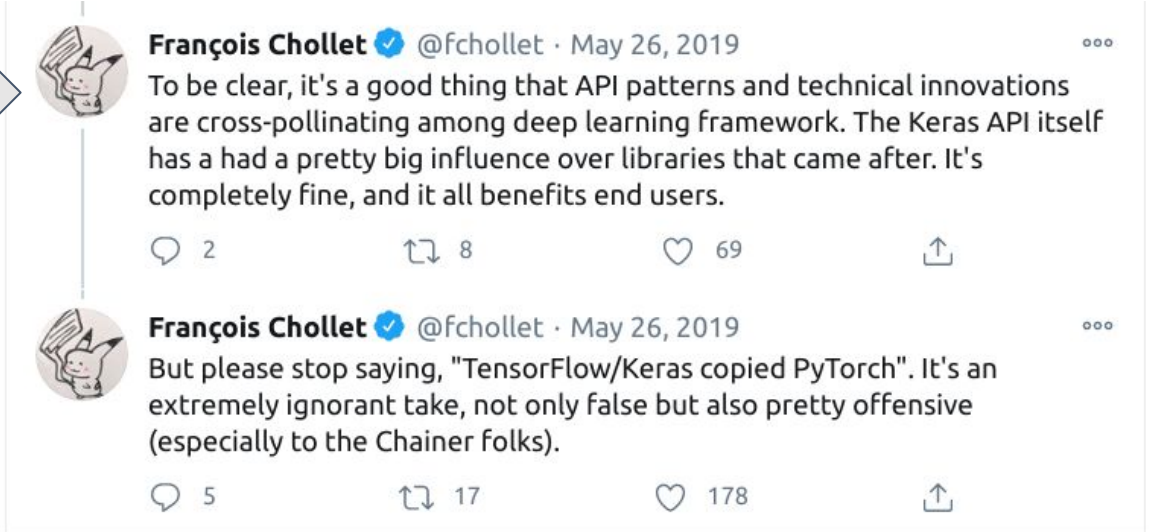
"I have felt like I have wasted years on tensorflow" - 2:25:52



Pytorch vs. tensorflow

Original creator of Keras





Mila

Consensus?

Use what makes you happy!

OpenAI

openai / jukebox

Code | Issues 90 | Pull requests 5 | Actions | Projects | Wiki

master | jukebox / jukebox / transformer / trar

prafullasd Merge pull request #61 from openai/more_sc

2 contributors

239 lines (214 sloc) | 10.7 KB

```
import functools
import numpy as np
import torch as t
import torch.nn as nn
import jukebox.utils.dist_adapter as dist
```

openai / gpt-2

Code | Issues 89 | Pull requests 29 | Actions | Security | Insights

master | gpt-2 / src / model.py | Jump to

WuTheFWasThat First commit

1 contributor

174 lines (144 sloc) | 6.35 KB

```
import numpy as np
import tensorflow as tf
from tensorflow.contrib.training import HParams
```

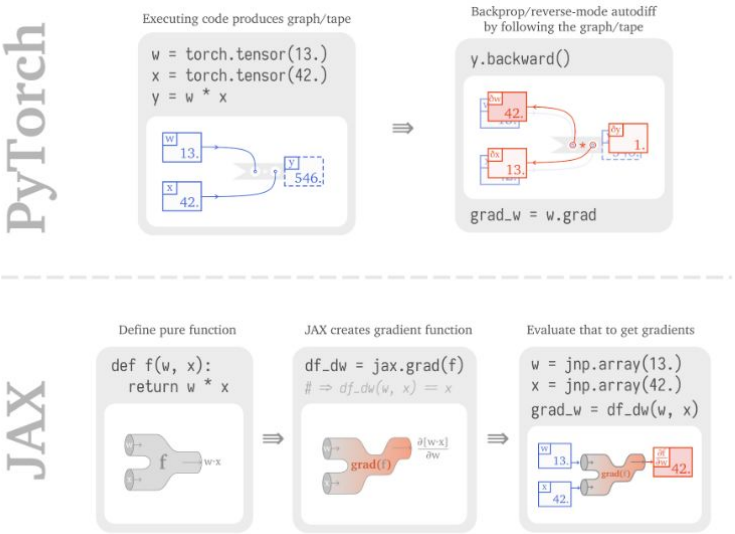
Mila

New kid on the block

(+Flax)

Very new + experimental, currently being used by google research on some new publications.

Use autodiff on **any** numpy function and use GPUs to compute it!



https://sjmielke.com/jax-purify.htm
https://github.com/google/jax
https://github.com/google-research/vision_transformer/blob/master/vit_jax/models.py

Mila

More Friends

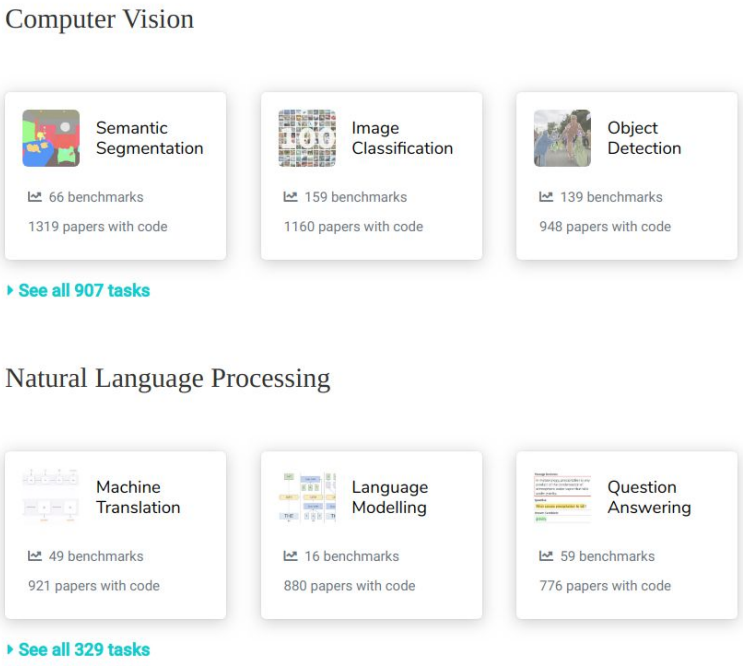
- **HuggingFace** has an extensive collection of **pretrained** language models and they are compatible across platforms (tf, pytorch)
- Very user-friendly API and useful implementations of tokenizers
- Support for dozens of languages
- Highly recommended for NLP projects

More Friends

Papers with code

Find the most recent papers, implementations and benchmarks.

“The mission of Papers with Code is to create a **free and open** resource with Machine Learning papers, code and evaluation tables.”



Mila

More Friends

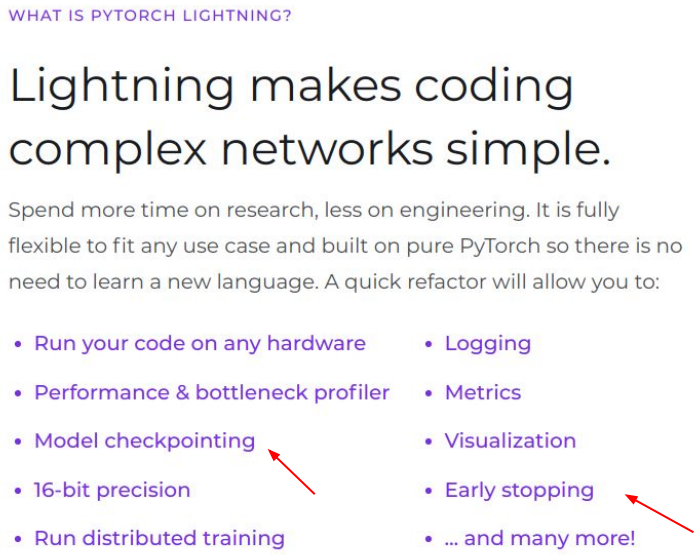
Pytorch lightning abstracts away the engineering:

- training loops
- Logging
- etc.

And lets you **focus** on the important stuff

- model creation,
- data processing
- etc.

Pytorch Lightning



A lot of these features come for free once you adapt your code to pytorch lightning.

A lot of these features are **error prone** when done in a DIY fashion

Pytorch Lightning makes it all standardized

Pytorch Lightning

WHAT IS PYTORCH LIGHTNING?

Lightning makes coding complex networks simple.

Spend more time on research, less on engineering. It is fully flexible to fit any use case and built on pure PyTorch so there is no need to learn a new language. A quick refactor will allow you to:

- Run your code on any hardware
- Performance & bottleneck profiler
- Model checkpointing
- 16-bit precision
- Run distributed training
- Logging
- Metrics
- Visualization
- Early stopping
- ... and many more!

Select GPU devices

You can select the GPU devices using ranges, a list of indices or a string containing a comma separated list of GPU ids:

```
# DEFAULT (int) specifies how many GPUs to use per node
Trainer(gpus=k)

# Above is equivalent to
Trainer(gpus=list(range(k)))

# Specify which GPUs to use (don't use when running on cluster)
Trainer(gpus=[0, 1])

# Equivalent using a string
Trainer(gpus='0, 1')

# To use all available GPUs put -1 or '-1'
# equivalent to list(range(torch.cuda.device_count()))
Trainer(gpus=-1)
```

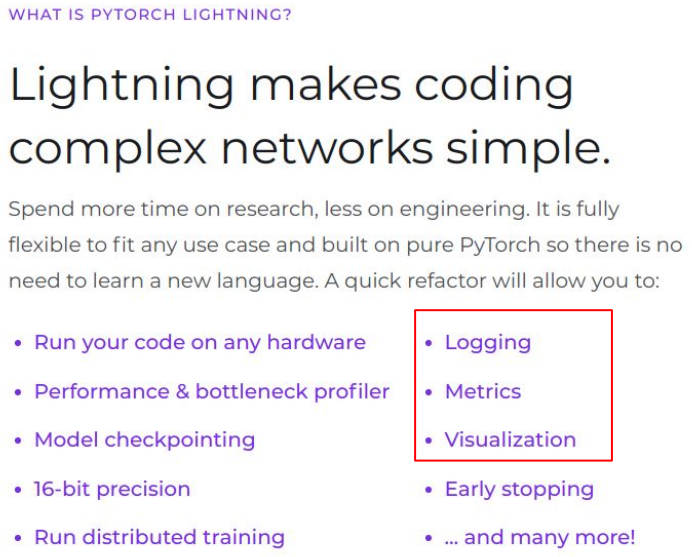
Distributed modes

Lightning allows multiple ways of training

- Data Parallel (*accelerator='dp'*) (multiple-gpus, 1 machine)
- DistributedDataParallel (*accelerator='ddp'*) (multiple-gpus across many machines (python script based)).
- DistributedDataParallel (*accelerator='ddp_spawn'*) (multiple-gpus across many machines (spawn based)).
- DistributedDataParallel 2 (*accelerator='ddp2'*) (DP in a machine, DDP across machines).
- Horovod (*accelerator='horovod'*) (multi-machine, multi-gpu, configured at runtime)
- TPUs (*tpu_cores=8|x*) (tpu or TPU pod)

https://pytorch-lightning.readthedocs.io/en/latest/multi_gpu.html

Pytorch Lightning



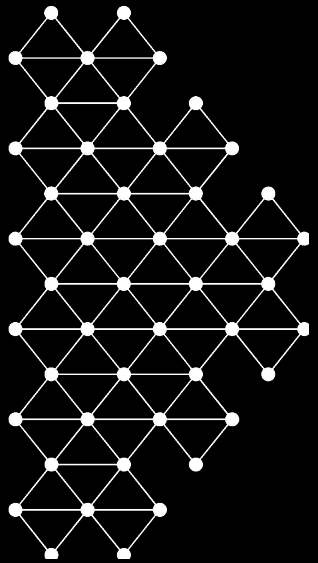
Logging with many libraries is supported out-of-the-box, seamlessly switch between them:

- MLFlow
- CometML
- Tensorboard
- Weights and Biases
- ...

Tensorbaord.dev and wandb ?

Quebec
Artificial
Intelligence
Institute

Mila



Software 2.0

Software 2.0

Software 1.0

- Code explicit instructions for a computer to execute
- Use human-readable language to program low-level machine instructions
- Python, C++, etc.

https://www.youtube.com/watch?v=DEqXNfs_HhY
https://www.a1k0n.net/2011/07/20/donut-math.html

https://medium.com/@karpathy/software-2-0-a64152b37c35

Software 2.0

Software 2.0

- “Human unfriendly” code - weights and biases of neural networks
- You (mostly) don’t program the low-level stuff - backprop, etc.
- You program the data, the loss function, monitor experiments, evaluate results, tweak the datasets

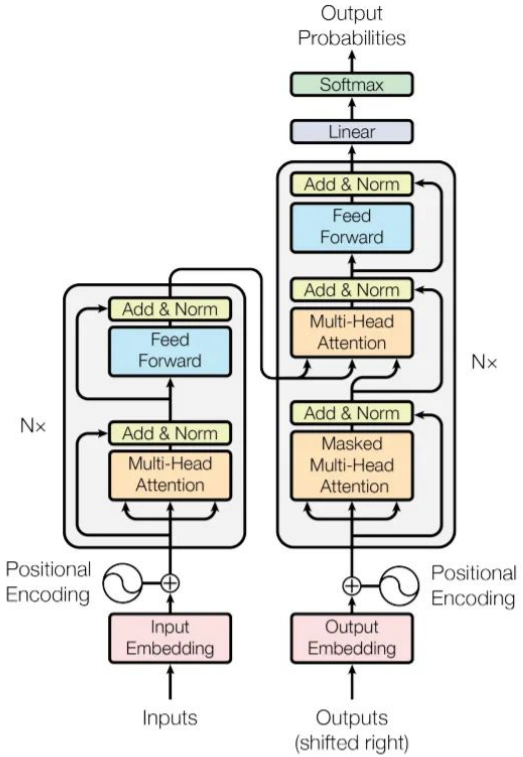
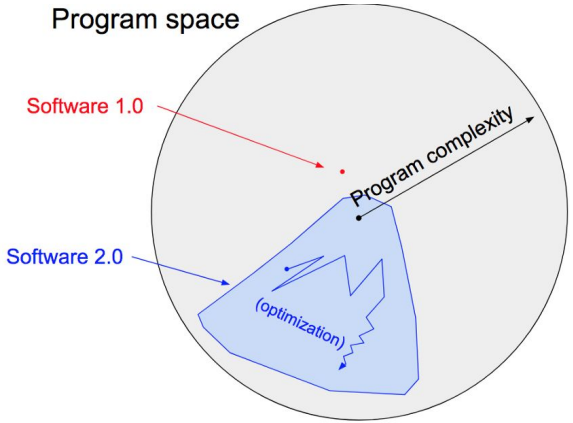


Figure 1: The Transformer - model architecture.

https://medium.com/@karpathy/software-2-0-a64152b37c35

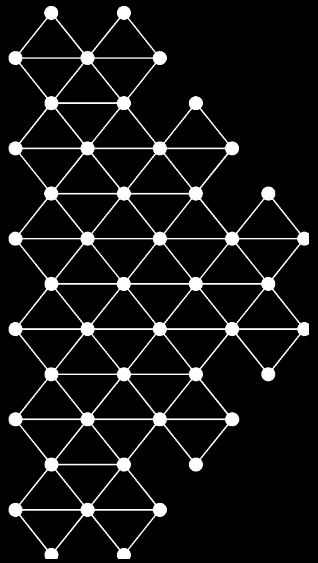
Software 2.0

- “[Software 2.0] identifies **a subset of program space** to search, and use the computational resources at our disposal to **search this space** for a program that works” - AKA gradient descent on neural networks



https://medium.com/@karpathy/software-2-0-a64152b37c35

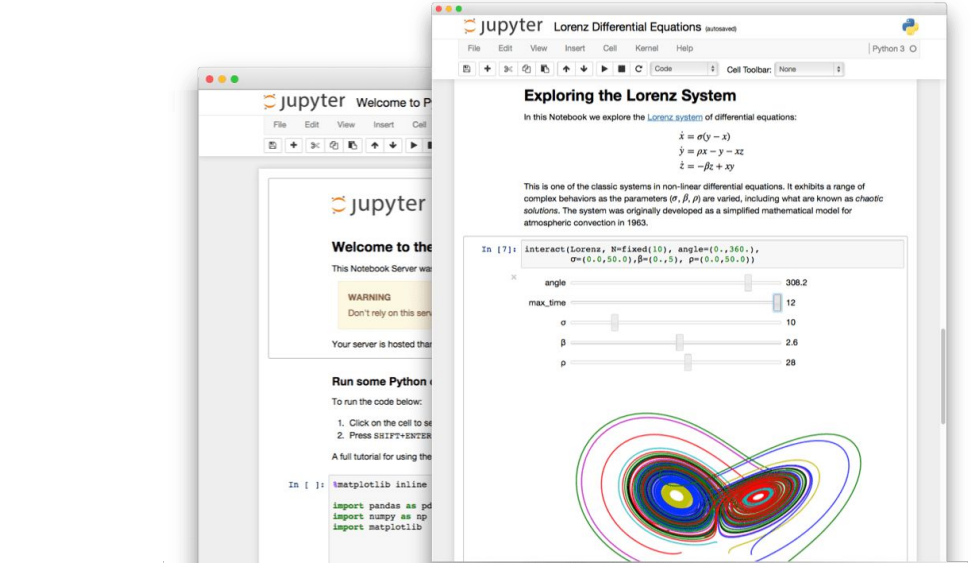
Quebec
Artificial
Intelligence
Institute



Notebooks

Interactive Notebooks

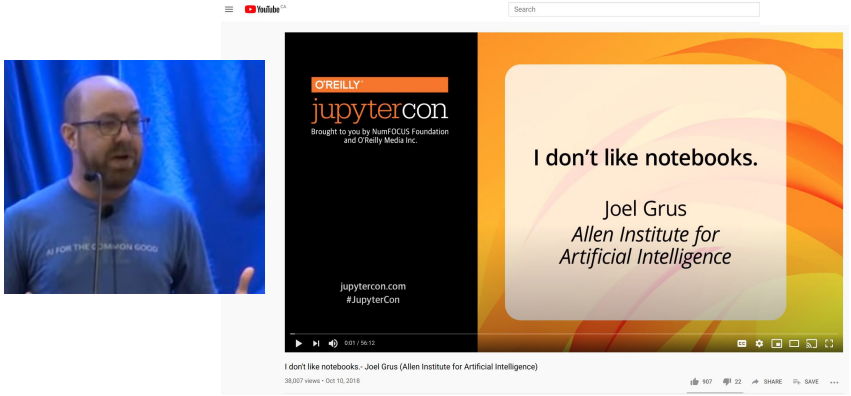
Interactive notebooks are a way to run python **code in cells** and view the generated output directly **inline**. They also support **markdown** to display inline text. It is very useful for generating **graphs**, displaying results, sharing ideas and rapid prototyping.



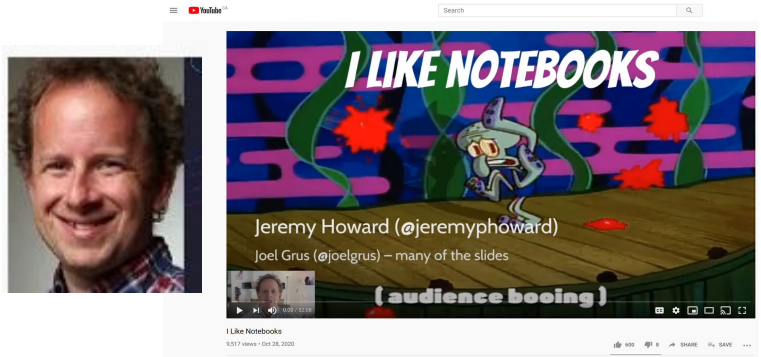
https://jupyter.org/

Notebooks: Pros and Cons

There are many great arguments about why notebooks are great and why sometimes they aren't.



https://youtu.be/7jiPeIFXb6U

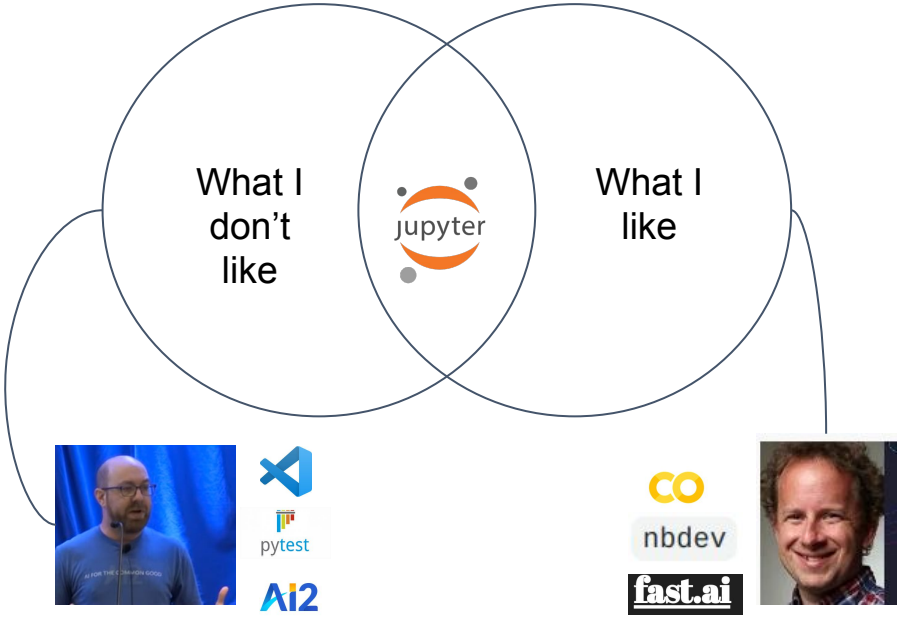


https://youtu.be/9Q6sLbz37gk

Notebooks: Pros and Cons

What they agree on:

- Notebooks are great for rapid prototyping and sharing ideas.
- Mixing markdown and code cells is really nice
- Hidden states are **DANGEROUS**
- Proper dev tools (CI, unit tests, etc.) are important

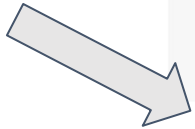


Notebooks: Danger zone

One common issue that arises with notebooks is that cells can be run in **arbitrary order**.

Example:

What do you expect the **accuracy** to be?



```
[1]  1 class MyBestModel():
     2   def fit(self, x_train, y_train):
     3     accuracy = 0.9
     4     return accuracy
     5
     6 class MyWorstModel():
     7   def fit(self, x_train, y_train):
     8     accuracy = 0.1
     9     return accuracy
```

```
[2]  1 model = MyBestModel()
```

```
[3]  1 x_train = [10, 1, 3, 3]
     2 y_train = [1, 0, 0, 0] # is digit odd or even?
     3 accuracy = model.fit(x_train, y_train)
     4 print("Model accuracy: ", accuracy)
```

```
Model accuracy:  ?
```

```
[4]  1 model = MyWorstModel()
```

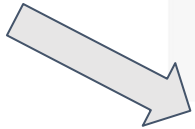
Notebooks: Danger zone

One common issue that arises with notebooks is that cells can be run in **arbitrary order**.

Example:

What do you expect the **accuracy** to be?

Nothing wrong here so far...



```
[1]  1 class MyBestModel():
     2   def fit(self, x_train, y_train):
     3     accuracy = 0.9
     4     return accuracy
     5
     6 class MyWorstModel():
     7   def fit(self, x_train, y_train):
     8     accuracy = 0.1
     9     return accuracy
```

```
[2]  1 model = MyBestModel()
```

```
[3]  1 x_train = [10, 1, 3, 3]
     2 y_train = [1, 0, 0, 0] # is digit odd or even?
     3 accuracy = model.fit(x_train, y_train)
     4 print("Model accuracy: ", accuracy)

     Model accuracy:  0.9
```

```
[4]  1 model = MyWorstModel()
```

Notebooks: Danger zone

Here the exact same notebook. No trickery was used. **What happened?**

```
[1]  1 class MyBestModel():
     2   def fit(self, x_train, y_train):
     3     accuracy = 0.9
     4     return accuracy
     5
     6 class MyWorstModel():
     7   def fit(self, x_train, y_train):
     8     accuracy = 0.1
     9     return accuracy
```

```
[2]  1 model = MyBestModel()
```

```
 1 x_train = [10, 1, 3, 3]
 2 y_train = [1, 0, 0, 0] # is digit odd or even?
 3 accuracy = model.fit(x_train, y_train)
 4 print("Model accuracy: ", accuracy)
--NORMAL--
Model accuracy:  0.1
```

```
[4]  1 model = MyWorstModel()
```

Notebooks: Danger zone

Here the exact same notebook. No trickery was used. **What happened?**

The cells were not run **in order**!

This happens way more than you might think.

```
[1]  1 class MyBestModel():
     2   def fit(self, x_train, y_train):
     3     accuracy = 0.9
     4     return accuracy
     5
     6 class MyWorstModel():
     7   def fit(self, x_train, y_train):
     8     accuracy = 0.1
     9     return accuracy
```

```
[2]  1 model = MyBestModel()
```

```
[5]  1 x_train = [10, 1, 3, 3]
     2 y_train = [1, 0, 0, 0] # is digit odd or even?
     3 accuracy = model.fit(x_train, y_train)
     4 print("Model accuracy: ", accuracy)
```

```
Model accuracy:  0.1
```

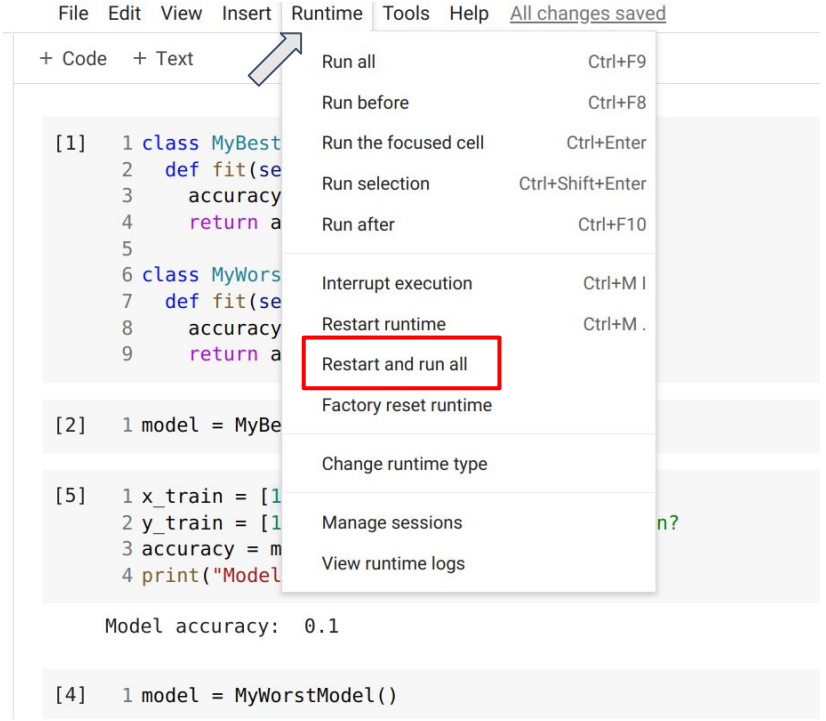
```
[4]  1 model = MyWorstModel()
```

Notebooks: Danger zone

Problems can also happen when **deleting** cells that had **global variables** being declared, making your code non-reproducible.

An easy way to avoid these mistakes is to **restart** and execute your notebooks regularly and **in order**!

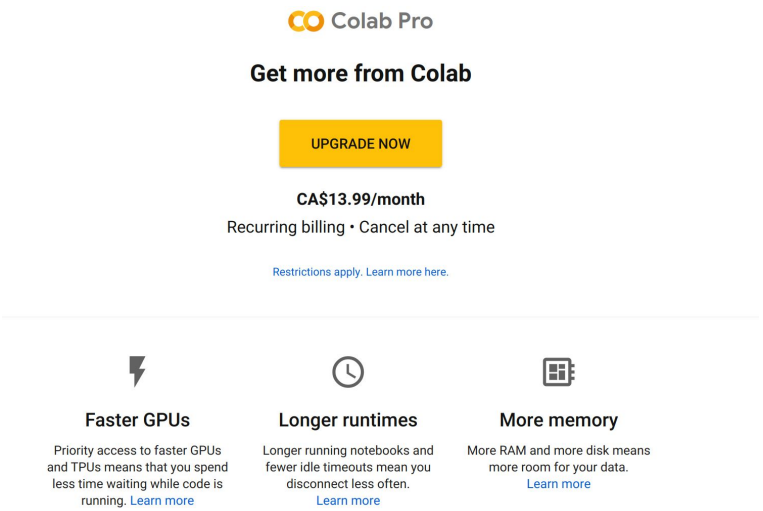
Always be sure that restarting and running all gives you the figures you expect.



Notebooks: cloud computing

You can now get priority access to GPU/TPUs when using Colab.

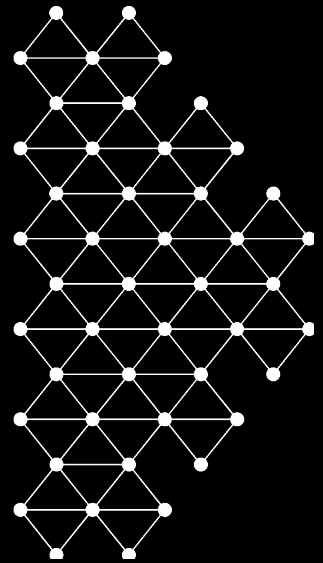
This is useful on **small-scale experiments** for GPU access at reduced costs.



DO NOT USE THIS FOR DEPLOYING PROJECTS!!!

- Instances can disconnect
- It does not scale

Quebec
Artificial
Intelligence
Institute



Deploying Models

Deploying Models

Once you have trained and validated a model, you need to **deploy** it to your user base. There are many different deployment strategies. We will look at two:

- Cloud deployment
- In-Browser deployment

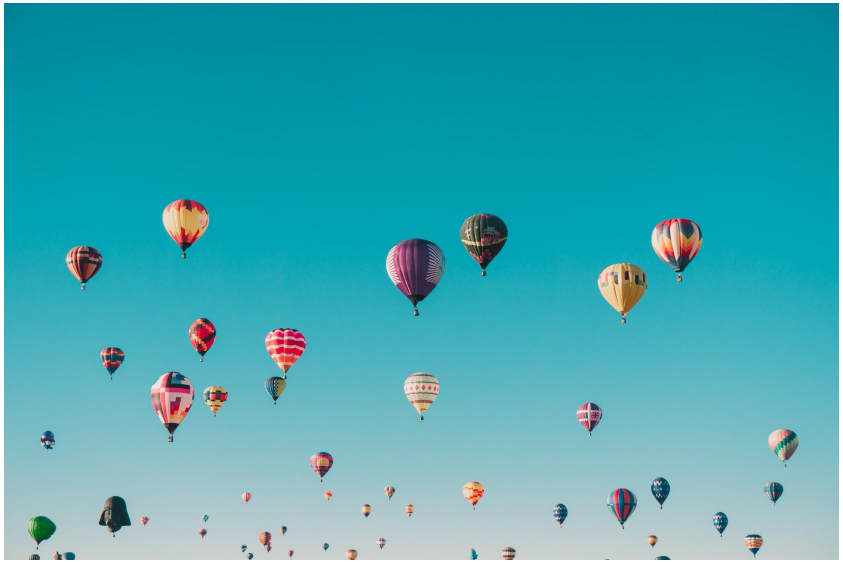
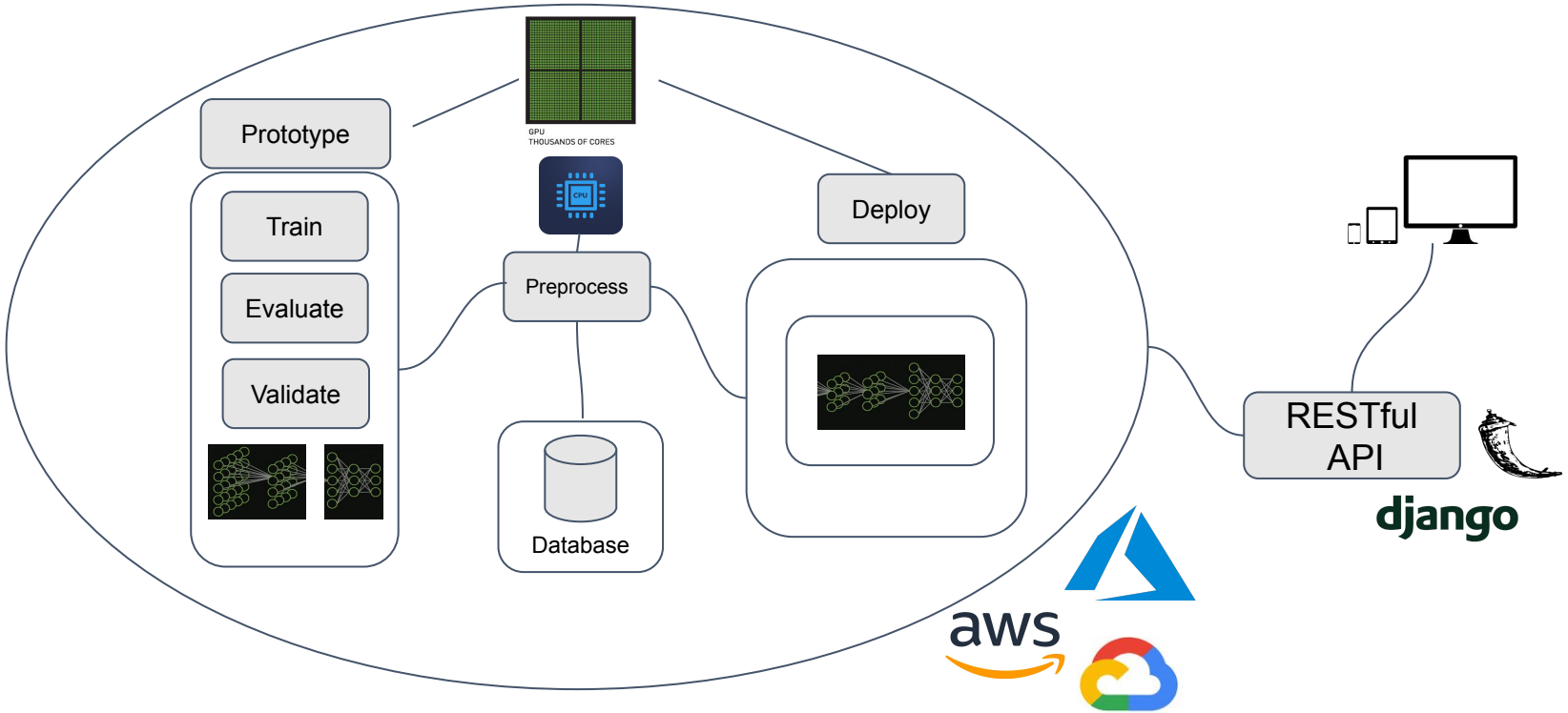


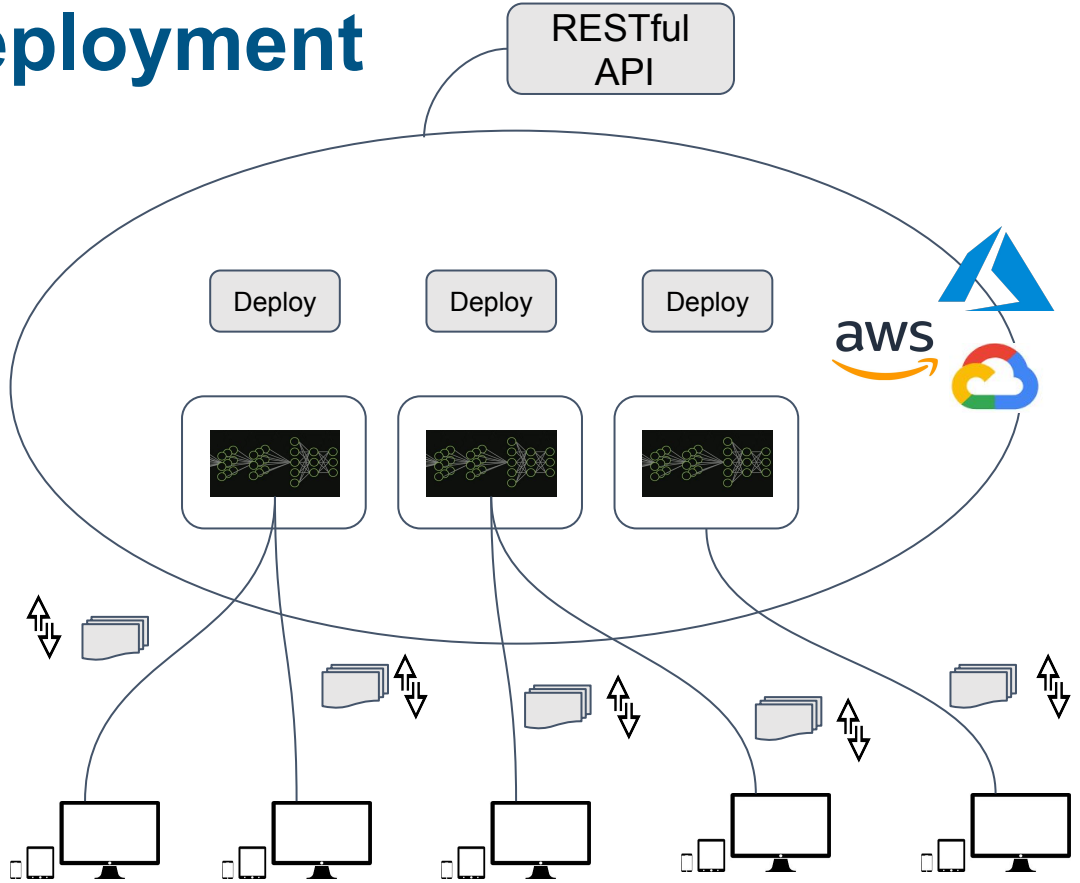
Photo by ian dooley on Unsplash

Cloud Computing Pipeline



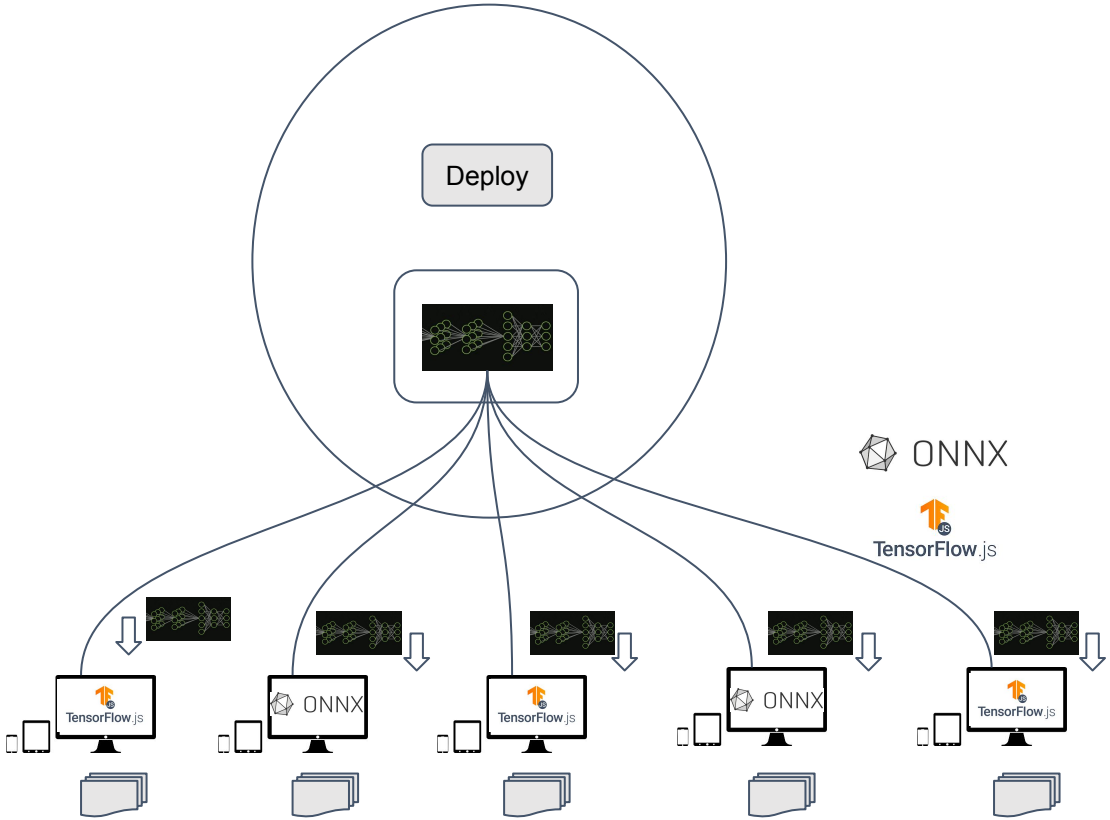
Option 1 - Cloud deployment

- Scale up and down based on demand
- Exchange data (inputs) for results (outputs)
- Faster Inference (for larger models + lots of data)
- **Online only**
- You own the models
- Less chances of things going wrong (control over compute)



Option 2 - In Browser

- Ship your model to clients
- Much less overhead - clients just download the model. Little hosting costs
- Your models are public
- Works offline
- Data can stay on client side
- Not adequate for large (>10 mb) models
- Inference time can be slow depending on hardware



[How to run PyTorch models in the browser with ONNX.js]