What is PyTorch

July 2022

AGENDA

01

WHAT IS PYTORCH

02

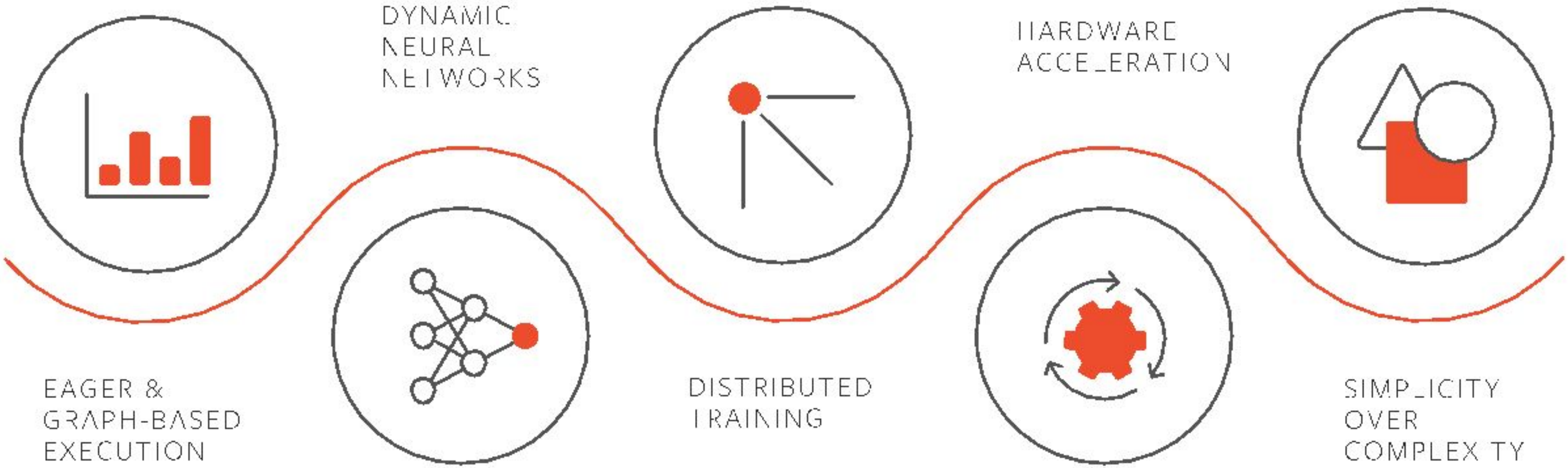
WHO MAKES PYTORCH

03

04

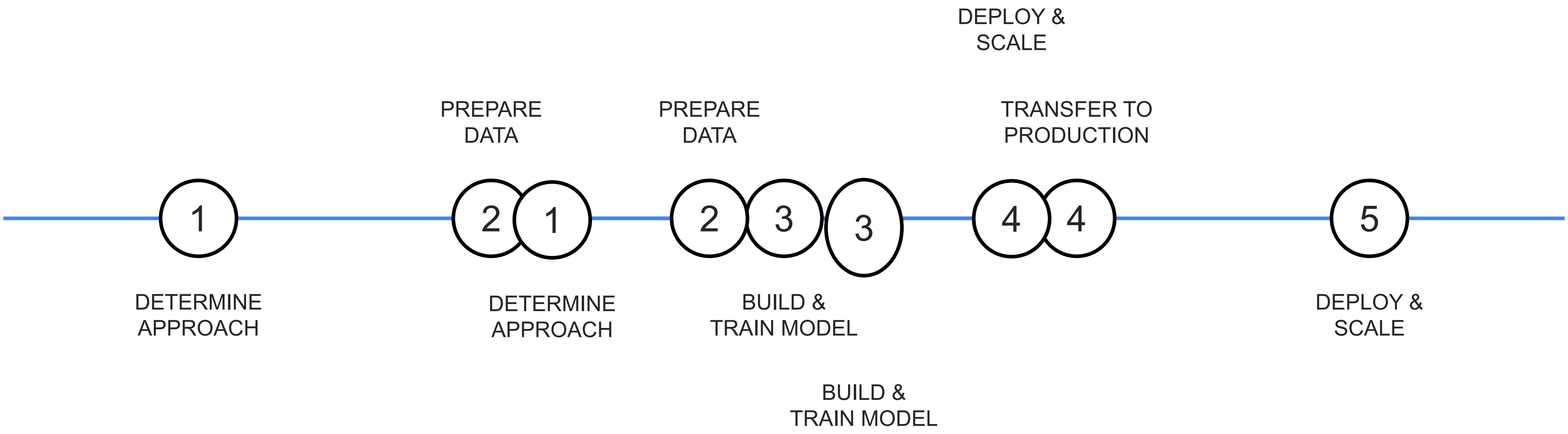
LEARN MORE

DESIGN PRINCIPLES & TENETS



PyTorch Mission

RESEARCH PROTOTYPING + PRODUCTION DEPLOYMENT


DEPLOY & SCALE
PREPARE DATA
PREPARE DATA
TRANSFER TO PRODUCTION
1
2
1
2
3
3
4
4
5
DETERMINE APPROACH
DETERMINE APPROACH
BUILD & TRAIN MODEL
DEPLOY & SCALE
BUILD & TRAIN MODEL

The Old Process

PROTOTYPING

PYTORCH

Research teams and engineers experiment with and train new algorithms and approaches.

TRANSFERRING

ONNX

Teams spend significant time porting models to run in large scale production environments.

DEPLOYING

Caffe2

Models are optimized and deployed in production using a separate high performance framework.

The New Process

PROTOTYPING

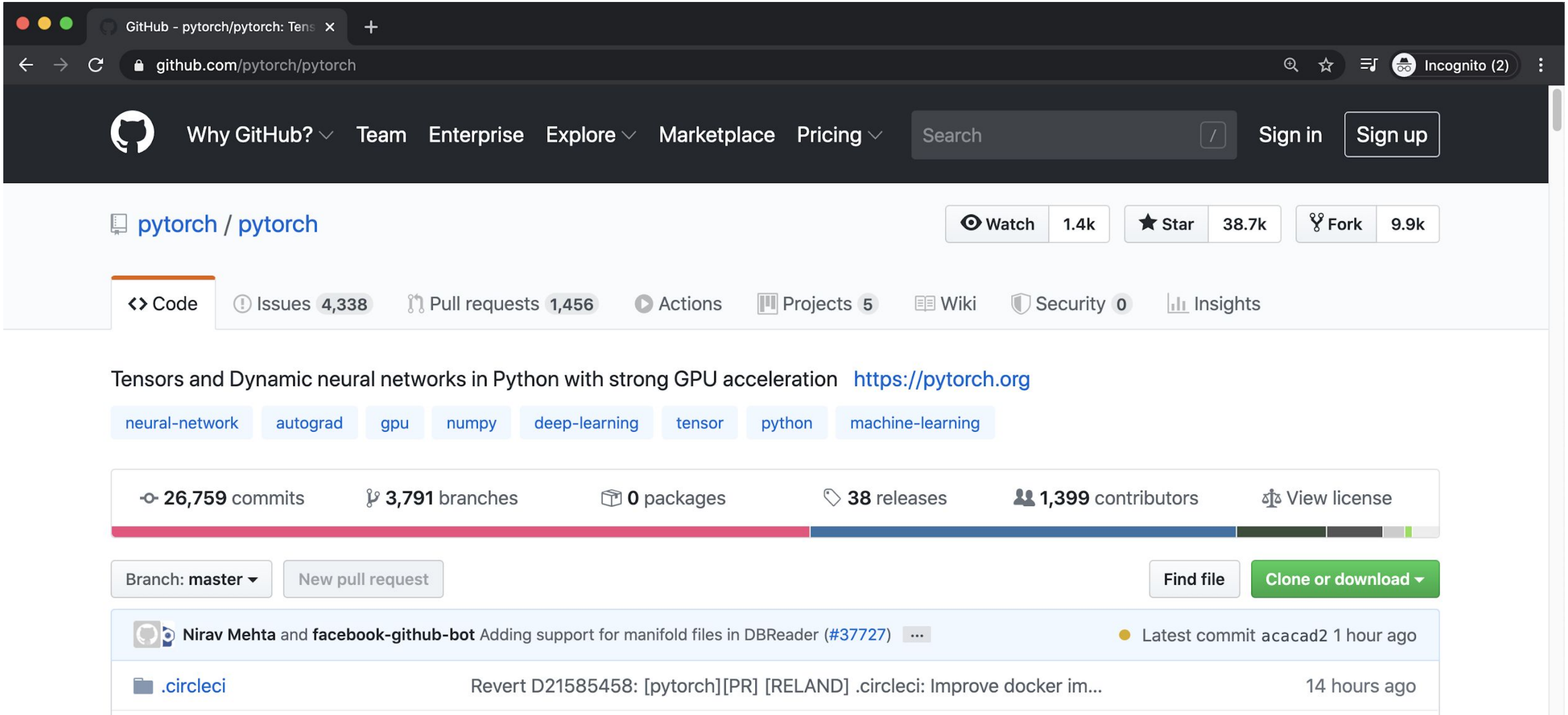
DEPLOYING

Experiment with algorithms and approaches and train new models

Use hybrid front-end to optimize models for production environment

Rapidly deploy proven models in production across devices

Open source, Community based Deep learning platform



WHO MAKES PYTORCH

THERE IS TREMENDOUS AMOUNT OF GROWTH YEAR OVER YEAR

1626	45K	34K
CONTRIBUTORS	DOWNSTREAM PROJECTS	PYTORCH FORUM USERS

2K+

CONTRIBUTORS

90K+

DEPENDENT PROJECTS ON GITHUB

1K+

FORUM TOPICS PER MONTH

PyTorch Mission

Alfredo Canziani
@alfcnz

Atcold/pytorch-Deep-Learning-Minicourse

ing with PyTorch. Contribute to
rning-Minicourse develpment
GitHub.

Smerity
@smerity

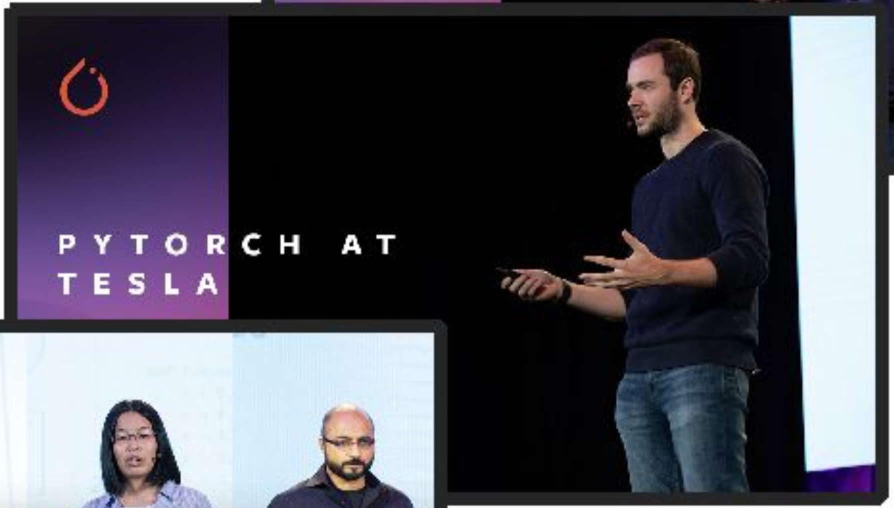
True to their mission, the @PyTorch community focused on solving the issues of eager mode w/o impacting
erability?
Want
ll without

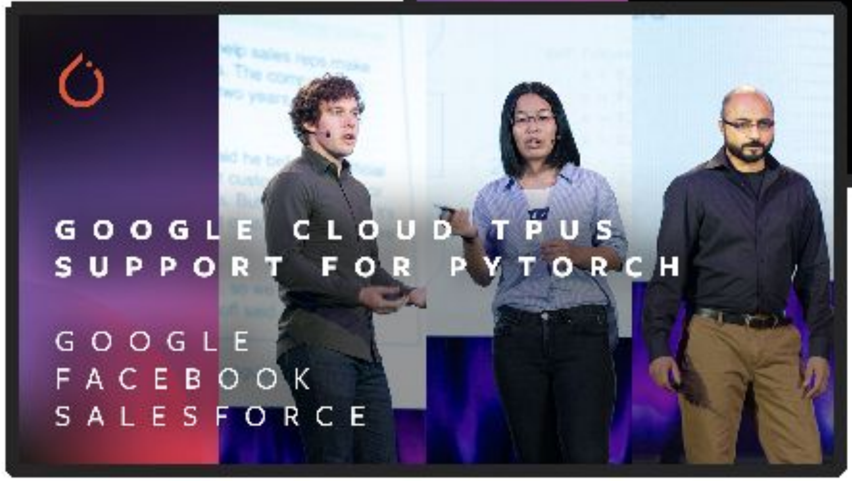
2 Oct 2018

Jeremy Howard
@jeremyphoward

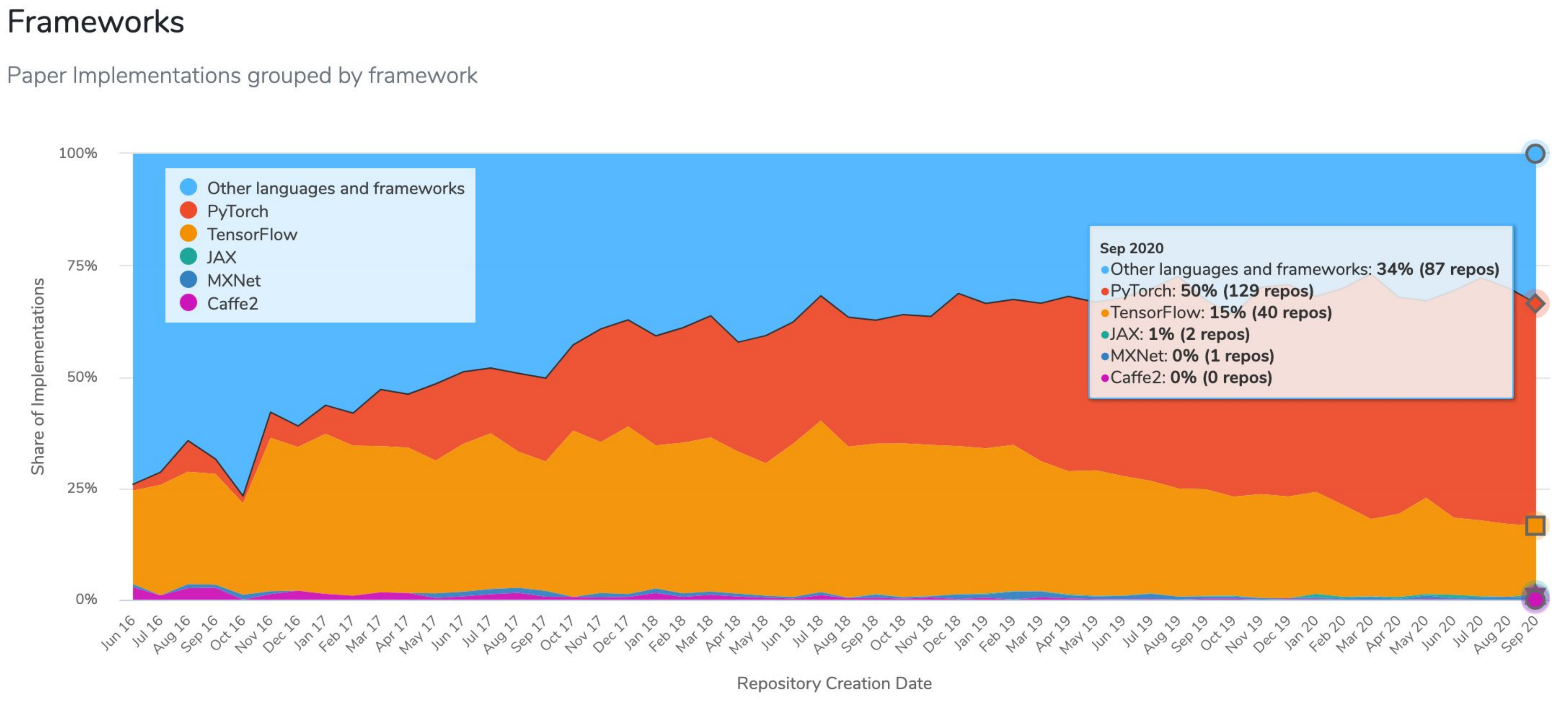
At the @PyTorch developer conference, I was part of a fascinating panel with @clattner_llvm, Yangqing Jia, and Noah Goodman, Expertly moderated by @soumithchintala. Here it is!

10 Oct 2018





PyTorch Community



Source: https://paperswithcode.com/trends

DEVELOPER EXPERIENCE

FULL FREEDOM TO ITERATE AND EXPLORE THE DESIGN SPACE

CLEAN AND INTUITIVE APIS

A RICH ECOSYSTEM OF TOOLS AND LIBRARIES

torch.nn

NEURAL NETWORKS

torch.optim

OPTIMIZERS

torch.data

DATASETS &
DATA LOADERS

torch.autograd

AUTO
DIFFERENTIATION

torch.vision

MODELS, DATA

torch.jit

TORCH SCRIPT
DEPLOYMENT

Thank you!

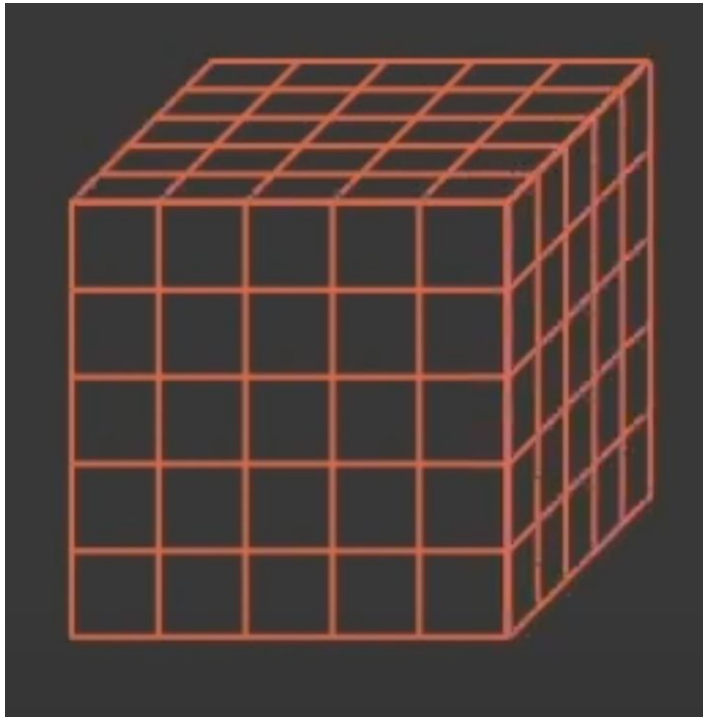
PyTorch Fundamentals

Tensors

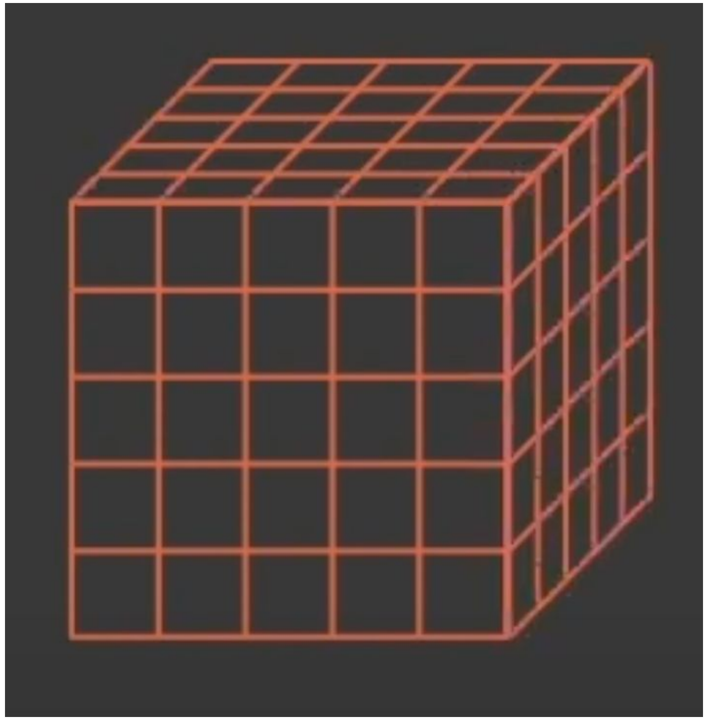
Autograd

High-level libraries

Tensors

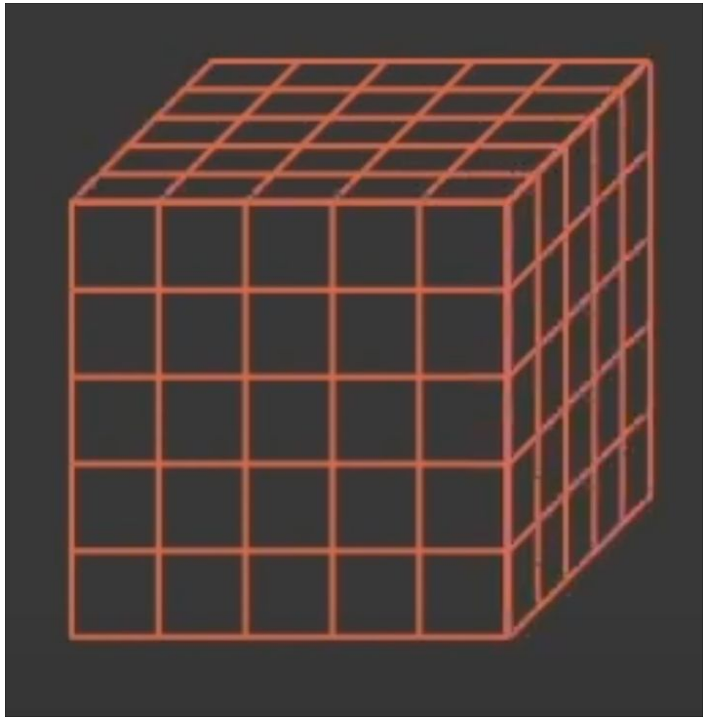


Tensors



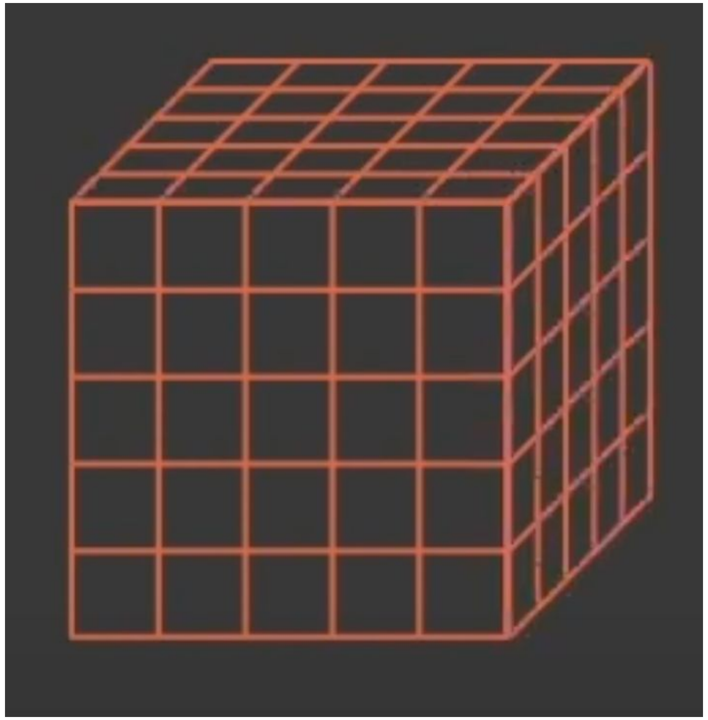
```
>>> tnsr = torch.tensor( [[2,3],[4,5]] )
```

Tensors



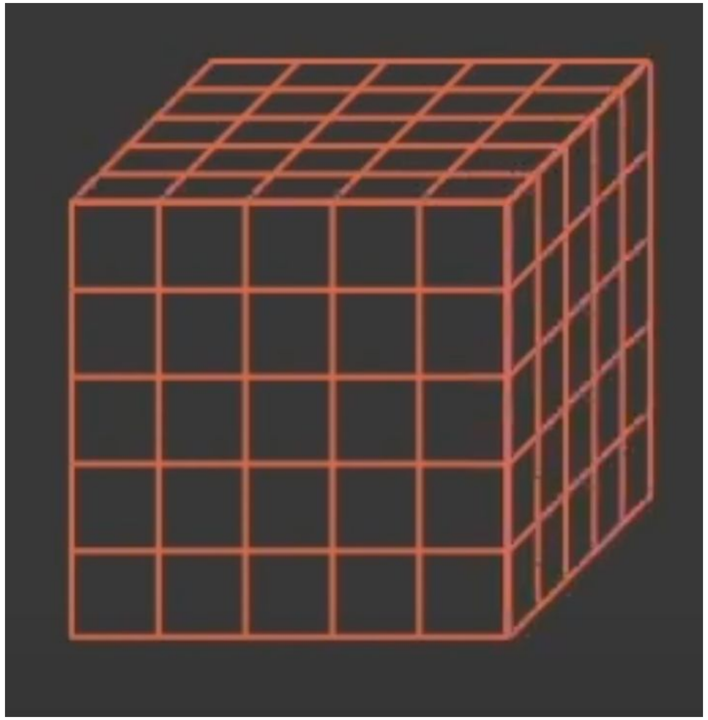
```
>>> tnsr = torch.tensor( [[2,3],[4,5]] )

>>> tnsr.shape
torch.Size([2, 2])
```

Tensors



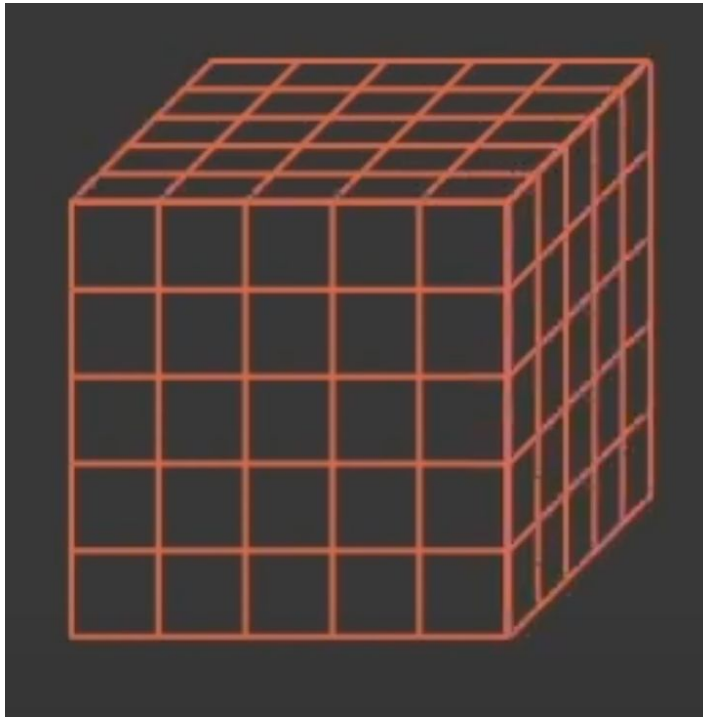
```
>>> tnsr = torch.tensor( [[2,3],[4,5]] )

>>> tnsr.shape
torch.Size([2, 2])

>>> tnsr.dtype
torch.float32
```

Tensors



```
>>> tnsr = torch.tensor( [[2,3],[4,5]] )

>>> tnsr.shape
torch.Size([2, 2])

>>> tnsr.dtype
torch.float32

>>> tnsr.to(‘cuda')
tensor([[2., 3.],
        [4., 5.]] , device='cuda:0')
```

Tensors



```
>>> tnsr = torch.tensor( [[2,3],[4,5]] )


>>> tnsr.shape
torch.Size([2, 2
```


```
>>> tnsr.dtype
torch.float32


>>> tnsr.to(‘cuda')
tensor([[2., 3.],
        [4., 5.]] , device='cuda:0')
```

Tensor API

TORCH

The torch package contains data structures for multi-dimensional tensors and mathematical operations over these are defined. Additionally, it provides many utilities for efficient serializing of Tensors and arbitrary types, and other useful utilities.

It has a CUDA counterpart, that enables you to run your tensor computations on an NVIDIA GPU with compute capability >= 3.0

Tensors

`is_tensor`	Returns True if *obj* is a PyTorch tensor.
`is_storage`	Returns True if *obj* is a PyTorch storage object.
`is_complex`	Returns True if the data type of `input` is a complex data type i.e., one of `torch.complex64`, and `torch.complex128`.
`is_floating_point`	Returns True if the data type of `input` is a floating point data type i.e., one of `torch.float64`, `torch.float32` and

Shared Memory Buffers

Bridge with numpy

CPU tensors and numpy arrays can share the same physical memory

```
np_arr = np.random.randn(4,4)
tnsr = torch.from_numpy(np_arr)
tnsr_view = tnsr.view(2,8)
```

```
[3] # Tensor and ndarray share the same underlying data buffer
    np_pointer = np_arr.__array_interface__['data'][0]
    tnsr_pointer = tnsr.storage().data_ptr()
    view_pointer = tnsr_view.storage().data_ptr()
```

```
[4] # Tensor and ndarray share the same underlying data buffer
    tnsr_pointer == np_pointer
```

```
[5] # Tensor and its view share the same underlying data buffer
    tnsr_pointer == view_pointer
```

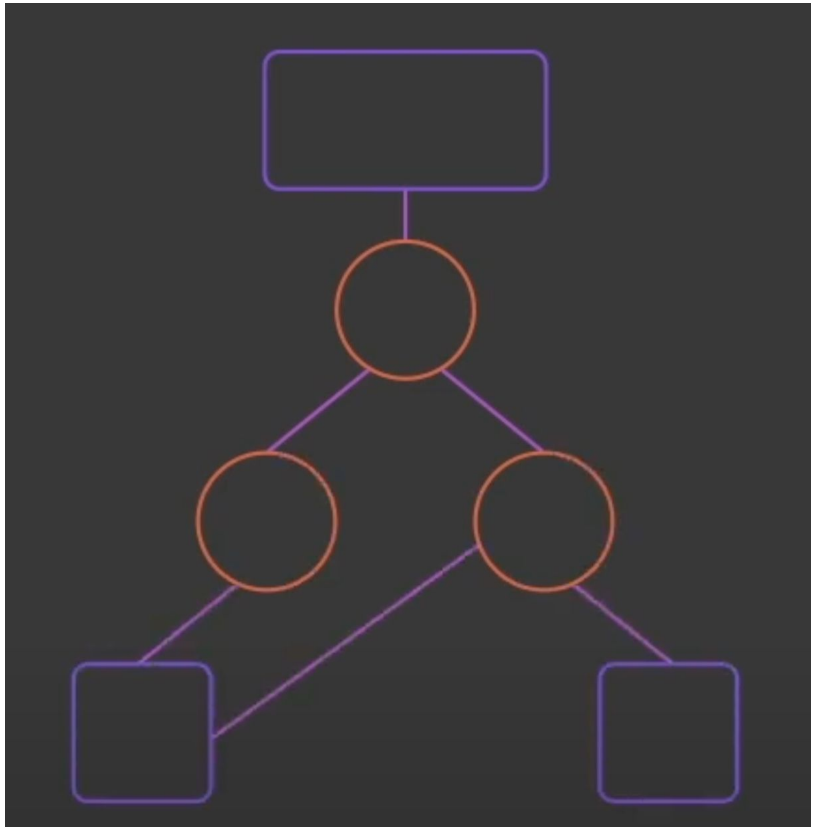
TL; DR

Tensors = numpy on steroids

Tons of operations on the GPU

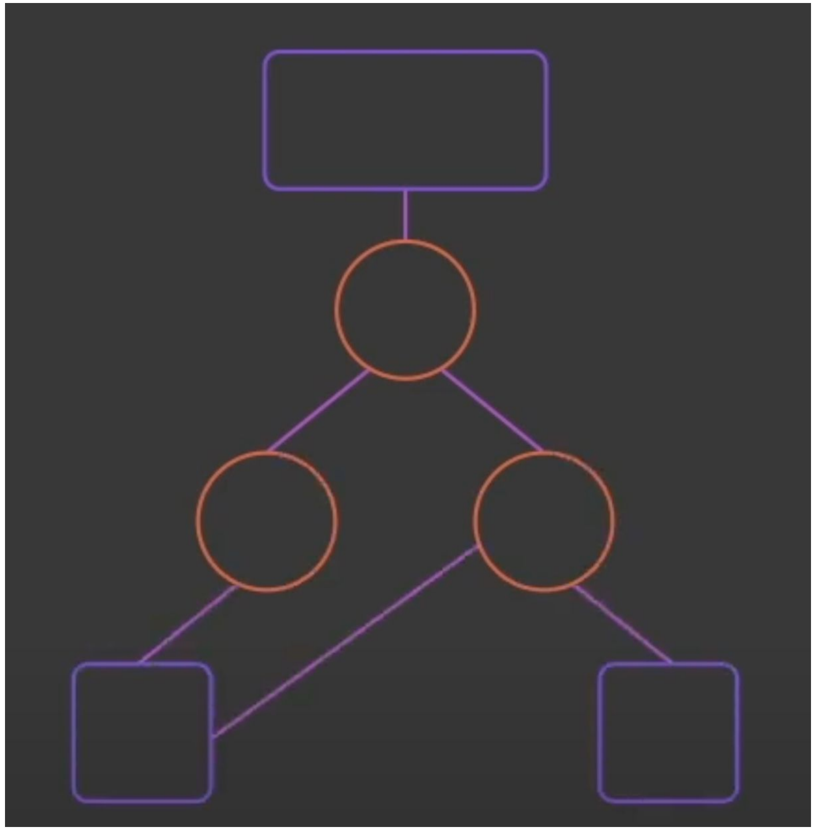
Super useful to store a NN’s parameters

Autograd



```
def train (error):
    while error > epsilon:
        backprop(error, params)
```

Autograd



```
def train (error):
    while error > epsilon:
        backprop(error, params)


def backprop (error, params):
    gradients = differentiate(error, params)
    update(params, gradients)
```

Autograd Example

```
[41] # Initialize "leaf" tensors that require_grad

     a = torch.tensor([2., 3.], requires_grad=True)
     b = torch.tensor([6., 4.], requires_grad=True)
```

$$Q = 3a^3 - b^2$$

```
[42] Q = 3*a**3 - b**2
     print(Q)

     Q.sum().backward() # convert vector to scalar before backpropagating
```

```
tensor([-12.,  65.], grad_fn=<SubBackward0>)
```

Autograd Example

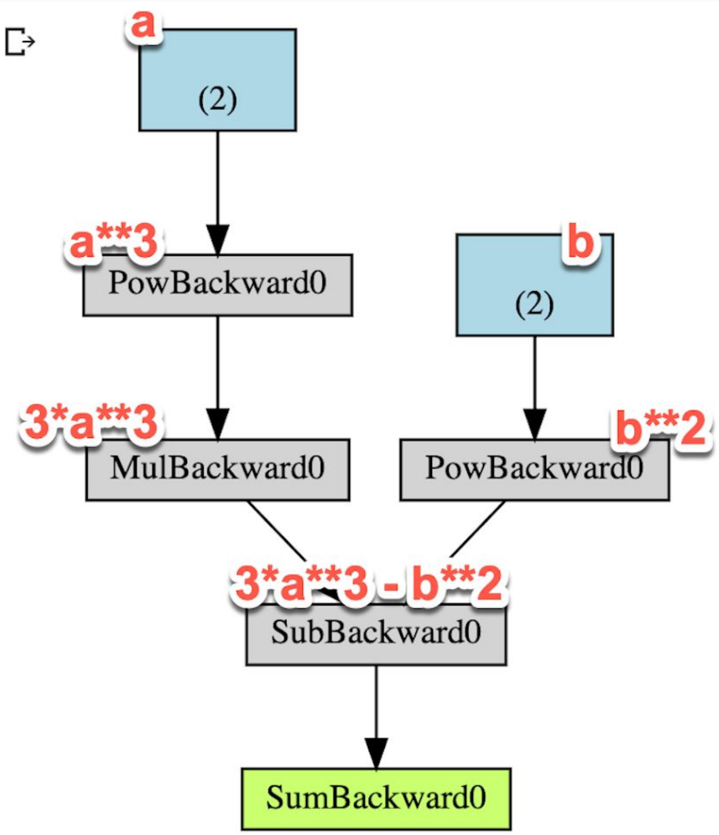
```
[41] # Initialize "leaf" tensors that require_grad

     a = torch.tensor([2., 3.], requires_grad=True)
     b = torch.tensor([6., 4.], requires_grad=True)
```

$$Q = 3a^3 - b^2$$

```
[42] Q = 3*a**3 - b**2
     print(Q)

     Q.sum().backward() # convert vector to scalar before backpropagating

tensor([-12.,  65.], grad_fn=<SubBackward0>)
```

Computational Graph

```
[40] torchviz.make_dot(Q.sum())
```



Autograd Example

```
[27] # Initialize "leaf" tensors that require_grad

     a = torch.tensor([2., 3.], requires_grad=True)
     b = torch.tensor([6., 4.], requires_grad=True)
```

$$Q = 3a^3 - b^2$$

```
[28] Q = 3*a**3 - b**2
     print(Q)
```

Higher-level libraries

1-layer NN from scratch

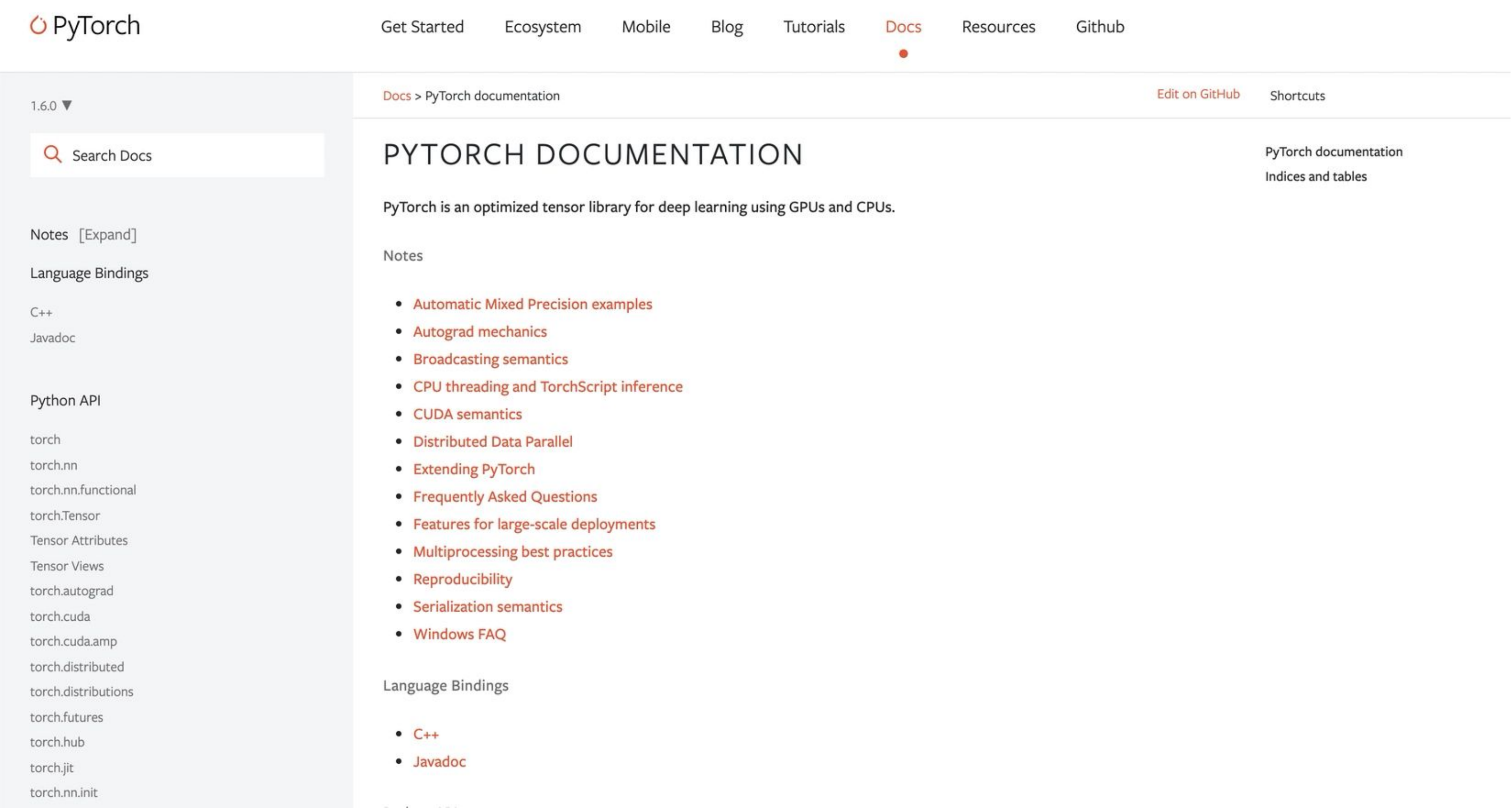
Parameters

```
[ ] import math

    weights = torch.randn(784, 10) / math.sqrt(784)
    weights.requires_grad_()
    bias = torch.zeros(10, requires_grad=True)
```

Operations

```
[ ] def log_softmax(x):
        return x - x.exp().sum(-1).log().unsqueeze(-1)

    def model(xb):
        return log_softmax(xb @ weights + bias)

    def nll(input, target):
        return -input[range(target.shape[0]), target].mean()

    def accuracy(out, yb):
        preds = torch.argmax(out, dim=1)
        return (preds == yb).float().mean()
```

Higher-level libraries



PyTorch

Get Started Ecosystem Mobile Blog Tutorials Docs Resources Github

1.6.0 ▼

Search Docs

Notes [Expand]

Language Bindings

C++
Javadoc

Python API

torch
torch.nn
torch.nn.functional
torch.Tensor
Tensor Attributes
Tensor Views
torch.autograd
torch.cuda
torch.cuda.amp
torch.distributed
torch.distributions
torch.futures
torch.hub
torch.jit
torch.nn.init

Docs > PyTorch documentation

Edit on GitHub

Shortcuts

PyTorch documentation
Indices and tables

PYTORCH DOCUMENTATION

PyTorch is an optimized tensor library for deep learning using GPUs and CPUs.

Notes

- Automatic Mixed Precision examples
- Autograd mechanics
- Broadcasting semantics
- CPU threading and TorchScript inference
- CUDA semantics
- Distributed Data Parallel
- Extending PyTorch
- Frequently Asked Questions
- Features for large-scale deployments
- Multiprocessing best practices
- Reproducibility
- Serialization semantics
- Windows FAQ

Language Bindings

- C++
- Javadoc

```python
net = Net()

data_loader = torch.utils.data.DataLoader(
    torchvision.datasets.MNIST('./data'))

optimizer = torch.optim.SGD(net.parameters())

for epoch in range(1, 11):
    for data, target in data_loader:
        optimizer.zero_grad()
        prediction = net.forward(data)
        loss = F.nll_loss(prediction, target)
        loss.backward()
        optimizer.step()
    if epoch % 2 == 0:
        torch.save(net, "net.pt")
```

PYTHON

```cpp
Net net;

auto data_loader = torch::data::data_loader(
  torch::data::datasets::MNIST("./data"));

torch::optim::SGD optimizer(net.parameters());

for (size_t epoch = 1; epoch <= 10; ++epoch) {
  for (auto batch : data_loader) {
    optimizer.zero_grad();
    auto prediction = net.forward(batch.data);
    auto loss = torch::nll_loss(prediction,
                                batch.label);
    loss.backward();
    optimizer.step();
  }
  if (epoch % 2 == 0) {
    torch::save(net, "net.pt");
  }
}
```

C++

TOOLS & LIBRARIES

Ax | BoTorch

PyTorch3D

CrypTen

Captum

PyTorch

Get Started | Ecosystem | Mobile | Blog | Tutorials | Docs | Resources | GitHub

ECOSYSTEM TOOLS

Tap into a rich ecosystem of tools, libraries, and more to support, accelerate, and explore AI development.

Join the Ecosystem

AdverTorch

A toolbox for adversarial robustness research. It contains modules for generating adversarial examples and defending against attacks.

Albumentations

Fast and extensible image augmentation library for different CV tasks like classification, segmentation, object detection and pose estimation.

Research to Production

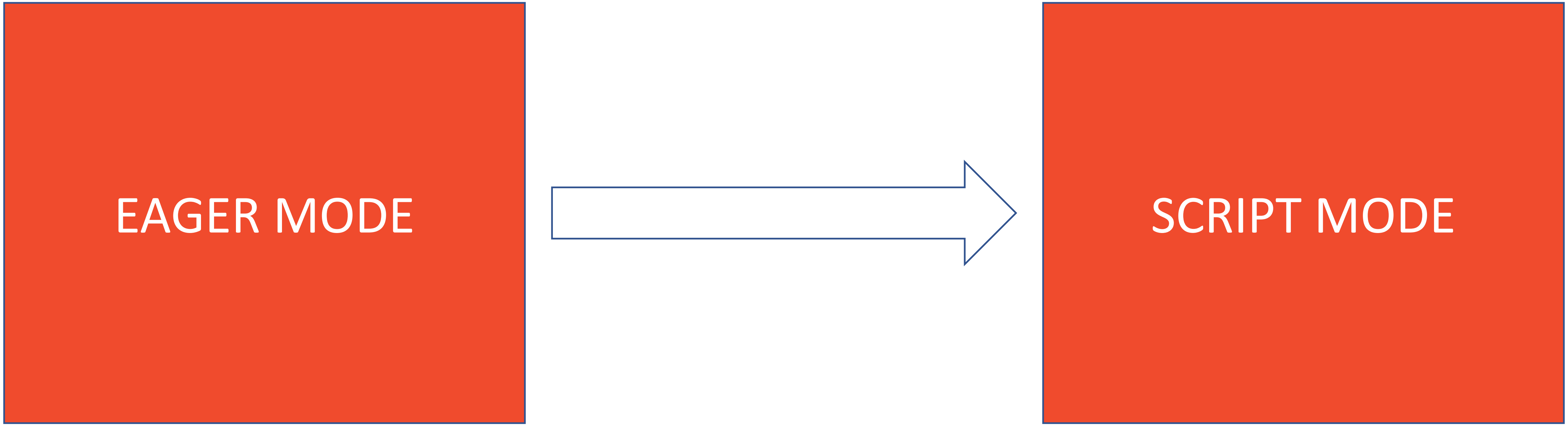
Portability and Performance

Embedded

- Training
- Inference
- Privacy
- Resource Constrained

At Scale

- Hardware Accelerated
- Continuously deployed
- Huge models
- Huge training jobs
- Batch and Synchronous inferences


EAGER MODE
SCRIPT MODE

PRUNING

State-of-the-art deep learning techniques rely on over-parametrized models that are hard to deploy.

Identify optimal techniques to compress models by reducing the number of parameters without sacrificing accuracy.

```
new_model = LeNet()
for name, module in new_model.named_modules():
    # prune 20% of connections in all 2D-conv layers
    if isinstance(module, torch.nn.Conv2d):
        prune.l1_unstructured(module, name='weight', amount=0.2)
    # prune 40% of connections in all linear layers
    elif isinstance(module, torch.nn.Linear):
        prune.l1_unstructured(module, name='weight', amount=0.4)

# to verify that all masks exist
print(dict(new_model.named_buffers()).keys())
```

QUANTIZATION

BETA

Neural networks inference is expensive

IoT and mobile devices have limited resources

Quantizing models enables efficient inference at scale

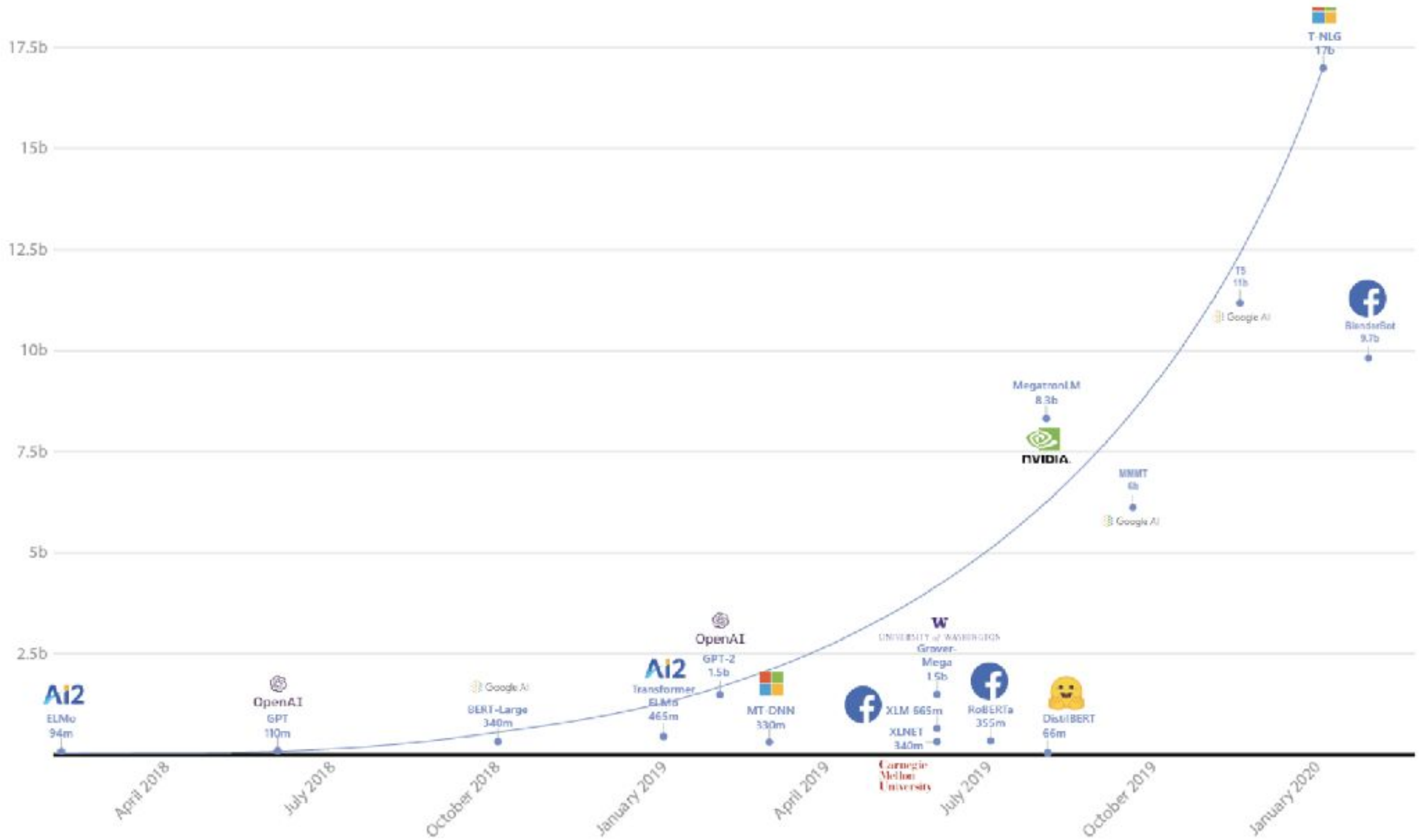
```
model = ResNet50()
model.load_state_dict(torch.load("model.pt"))

qmodel = quantization.prepare(
        model, {"": quantization.default_qconfig})

qmodel.eval()
for batch, target in data_loader:
        model(batch)

qmodel = quantization.convert(qmodel)
```

EXAMPLE MODELS | W/ QUANTIZATION

	fp32 accuracy	int8 accuracy change	Technique	CPU inference speed up
ResNet50	76.1 Top-1, Imagenet	-0.2 75.9	Post Training	2x 214ms → 102ms, Intel Skylake-DE
MobileNetV2	71.9 Top-1, Imagenet	-0.3 71.6	Quantization-Aware Training	4x 75ms → 18ms OnePlus 5, Snapdragon 835
Translate / FairSeq	32.78 BLEU, IWSLT 2014 de-en	0.0 32.78	Dynamic (weights only)	4x for encoder Intel Skylake-SE

These models and more available on TorchHub - https://pytorch.org/hub/

PARAMETER COUNTS CONTINUE TO GROW



REALLY LARGE SCALE

Tens of billions of training examples

Some models larger than size of machine

Hundreds of experimental runs competing for resources

More than 1000+ different production models

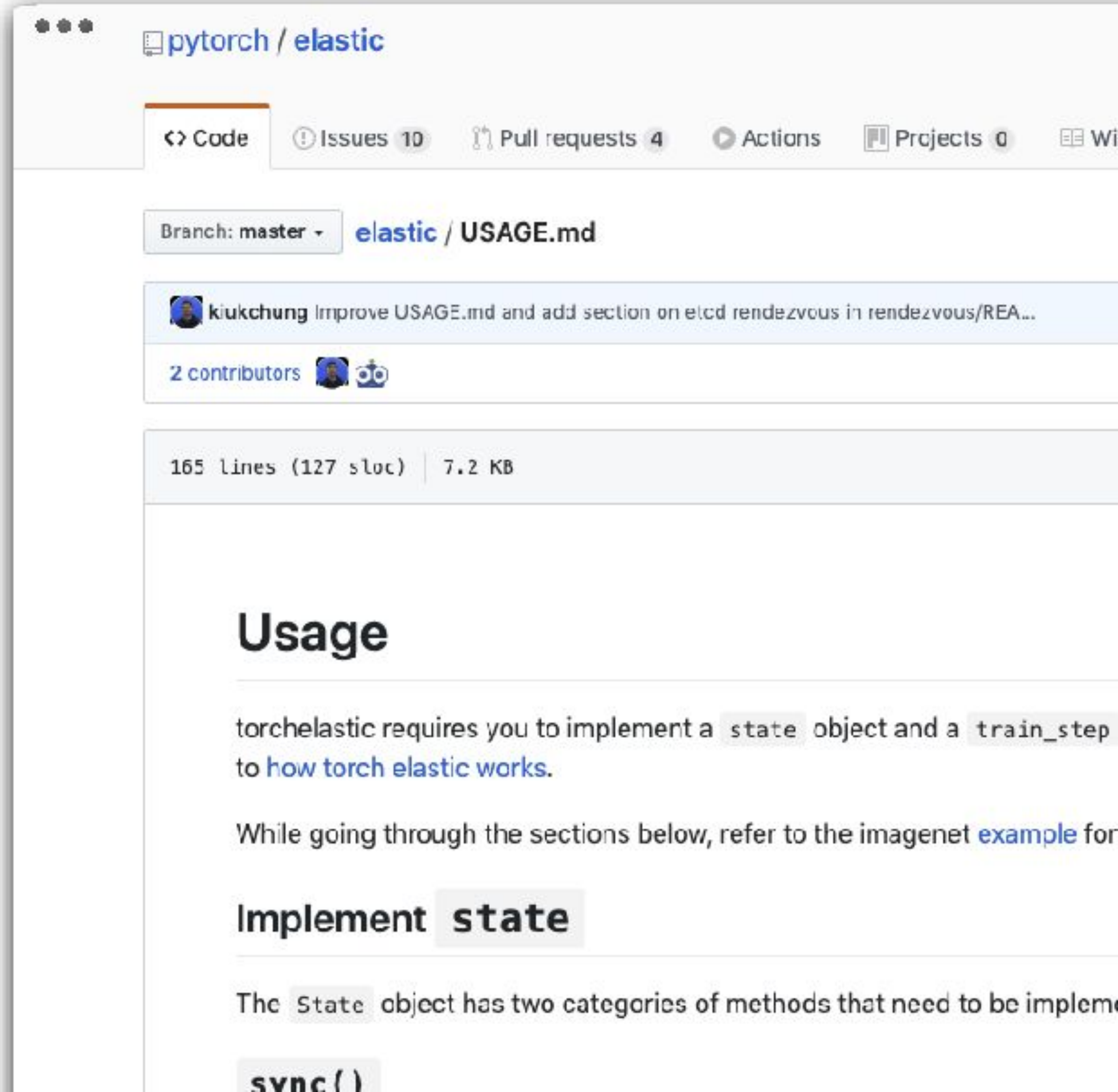
PYTORCH ELASTIC

Enables users to write fault tolerant and elastic distributed PyTorch jobs

Use cases

- *Fault tolerance*
  - Run on non-HPC cluster (e.g. cloud)
  - Mission critical production job
- *Dynamic Capacity Management:*
  - Run on leased capacity that can be preempted (e.g. AWS spot instances)
  - Shared pools where the pool size can change dynamically based on demand (e.g. autoscaling)

Open sourced 0.2.0 - https://github.com/pytorch/elastic



PYTORCH RPC (REMOTE PROCEDURE CALL)

Enables applications to run functions remotely, and automatically handles autograd if necessary.

Use cases

Scale out applications

Parameter Server Framework

In RL, multiple observers streaming states and rewards to the agent for policy training

Distributed model parallelism

Allow large models to span multiple machines

Hogwild! training

PYTORCH RPC COMPONENTS

RPC

Run user code with given args on the specified destination

Remote Reference (RRef)

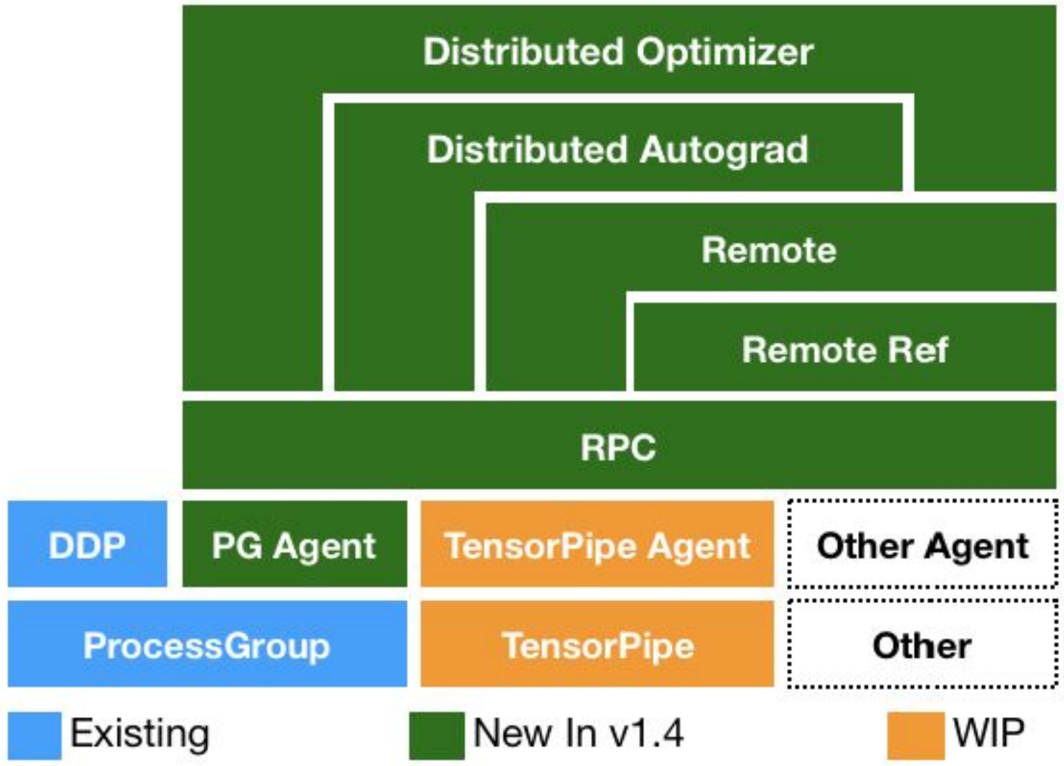
Tracks and maintains objects owned by a remote worker.

Distributed Autograd

Connects autograd graphs on different workers into one global graph and provides a similar backward API.

Distributed Optimizer

Automatically handles RRef of subnets and expose a similar API as local optimizers.



INFERENCE AT SCALE

Deploying and managing models in production is difficult.

Some of the pain points include:

Loading and managing multiple models, on multiple servers or end devices

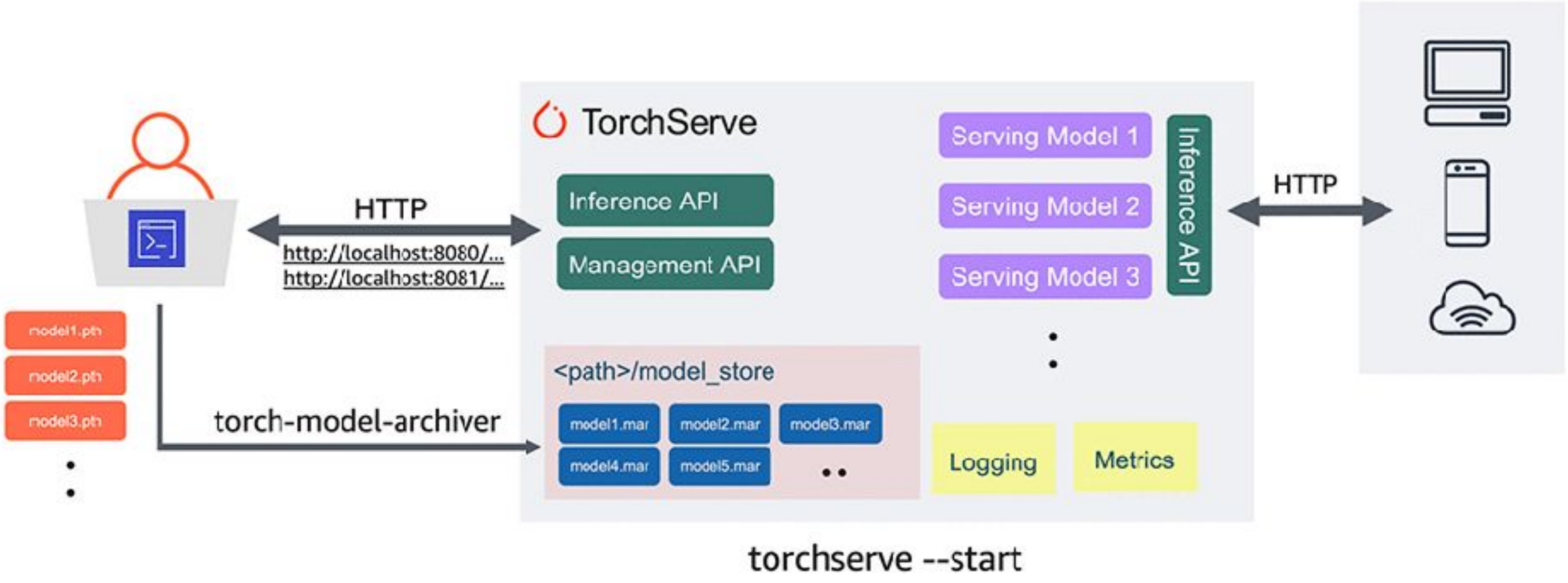
Running pre-processing and post-processing code on prediction requests.

How to log, monitor and secure predictions

What happens when you hit scale?

TORCHSERVE

BETA



- Default handlers for common use cases (e.g., image segmentation, text classification) along with custom handlers support for other use cases
- Model versioning and ability to roll back to an earlier version
- Automatic batching of individual inferences across HTTP requests
- Logging including common metrics, and the ability to incorporate custom metrics
- Robust HTTP APIS - Management and Inference

JOIN THE PYTORCH DEVELOPER COMMUNITY

PyTorch.org

Twitter.com/pytorch

Youtube.com/pytorch

Facebook.com/pytorch

Medium.com/pytorch