

PyTorch 1.0

Bringing Research and
Production Together

Dmytro Dzhulgakov

ENGINEERING LEAD | FACEBOOK AI



Agenda

1

AI @
Facebook

2

PyTorch Tech
Deep Dive

3

Bridging
Research and
Production

4

PyTorch
Ecosystem





ENHANCING EXISTING PRODUCTS

SOCIAL RECOMMENDATIONS

Carson Cohen looking for

Anna Shelley
This restaraunt is amazing!

Avista Restaurant
4.9 ★ · Open Now · \$\$\$ · Italian din ...
E Santa Clara St · (415) 737-3193

Kevin Lee
Check this out

Red Dragon
4.8 ★ · Open Now · \$\$ · Asian Rest ...
89 Mission St · (415) 510-4117

Molly Hakes and 27 others · 18 Comments

Carson Davis
This is my favorite one

The Fusion Market
5.0 ★ · Open Now · \$\$ · Casual choic ...
37 Mission St · (510) 625-6133

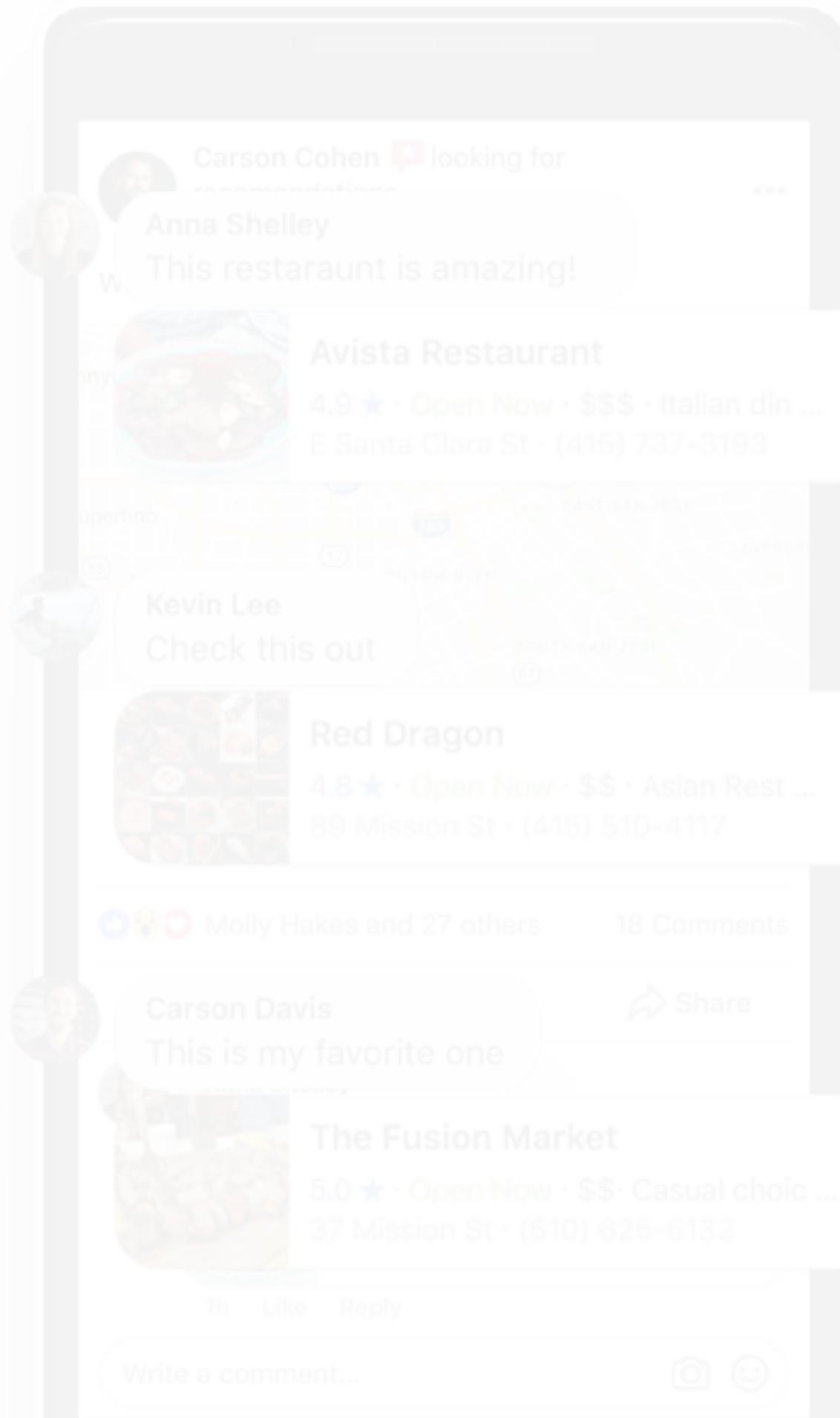
Write a comment...

A screenshot of a mobile application interface displaying a social recommendation feed. The feed consists of three separate card-like entries, each containing a user's profile picture, their name, a short comment, and a detailed card for a specific restaurant. The first entry is from 'Anna Shelley' who says 'This restaraunt is amazing!' and recommends 'Avista Restaurant'. The second entry is from 'Kevin Lee' who says 'Check this out' and recommends 'Red Dragon'. The third entry is from 'Carson Davis' who says 'This is my favorite one' and recommends 'The Fusion Market'. Each restaurant card includes a thumbnail image, its rating, status ('Open Now'), price range ('\$\$\$' or '\$\$'), cuisine type ('Italian din ...' or 'Asian Rest ...'), address, and phone number.



ENHANCING EXISTING PRODUCTS

SOCIAL RECOMMENDATIONS



MACHINE TRANSLATIONS

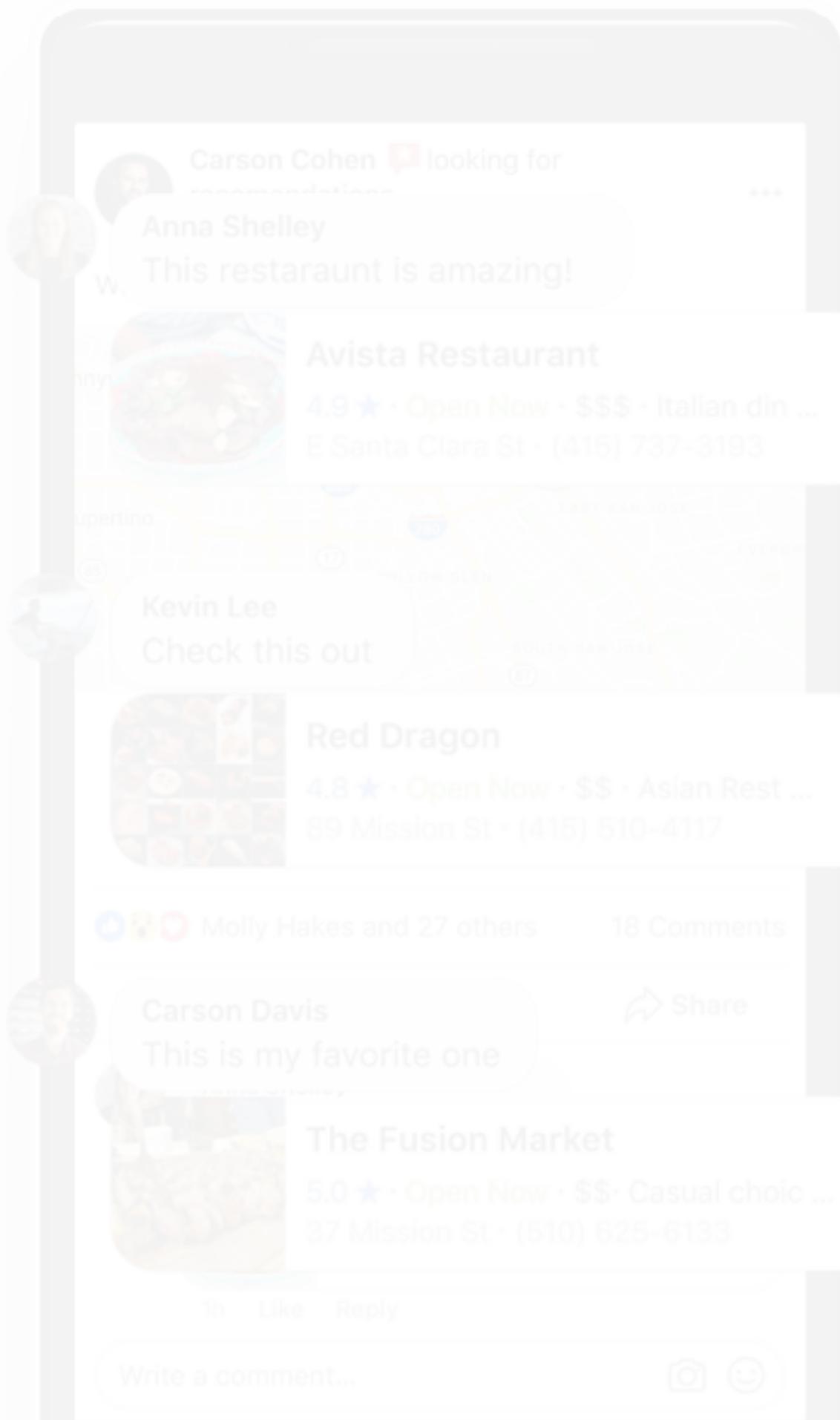


- 45+: Languages supported for translation
- 2K+: Translation directions
- 6B+: Translation impressions per day

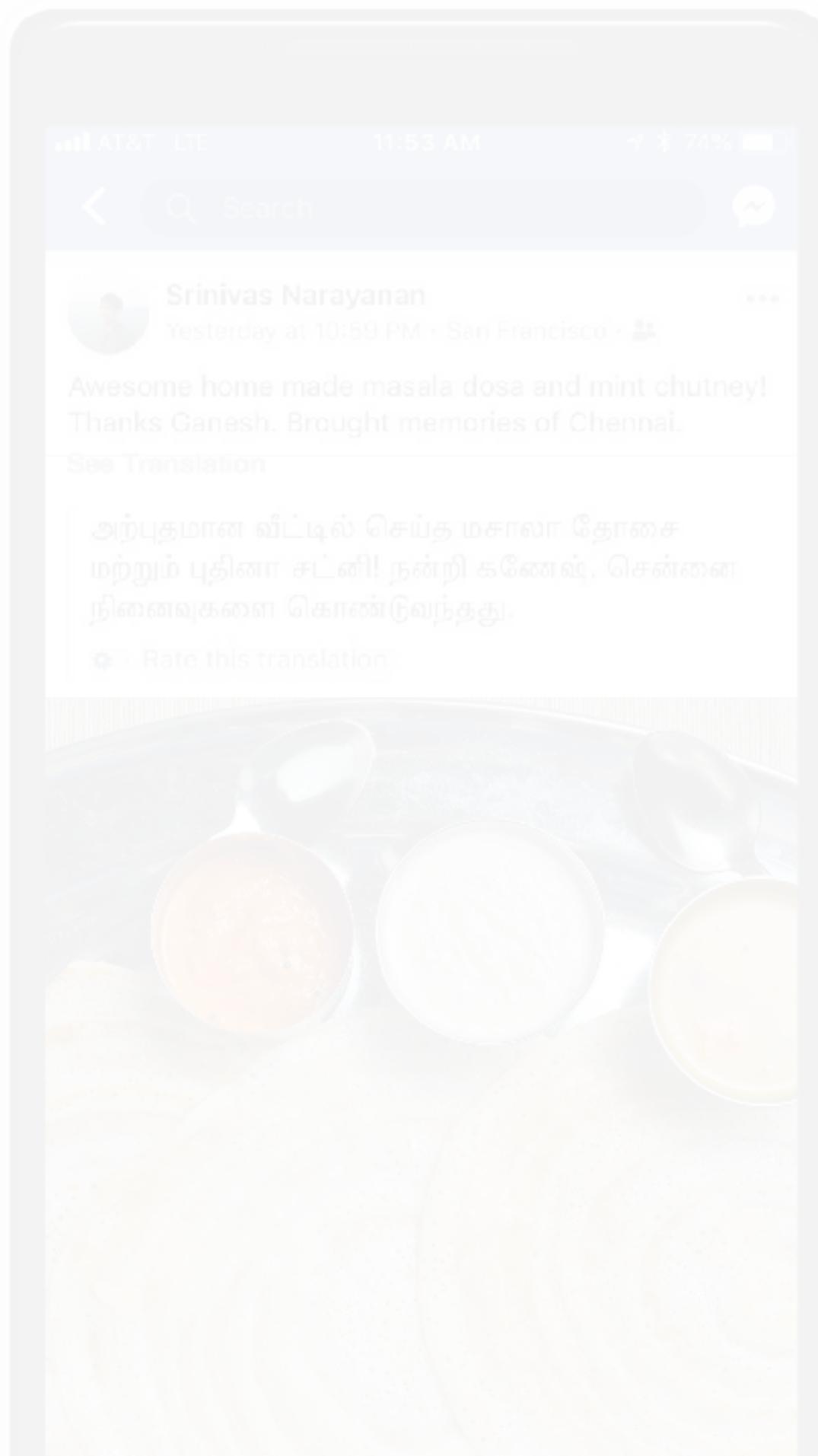


ENHANCING EXISTING PRODUCTS

SOCIAL RECOMMENDATIONS



MACHINE TRANSLATIONS



ACCESSIBILITY

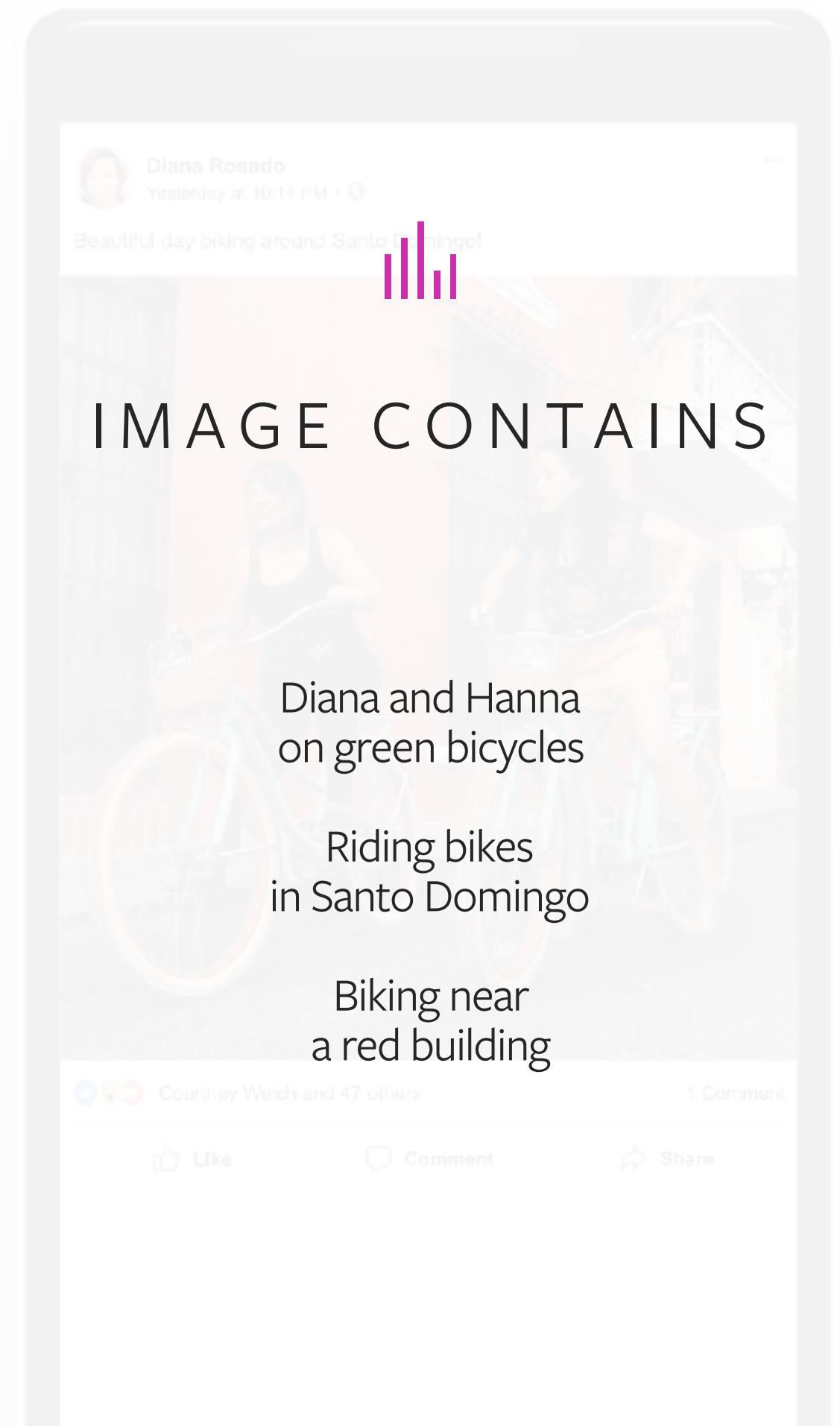


IMAGE CONTAINS

Diana and Hanna
on green bicycles

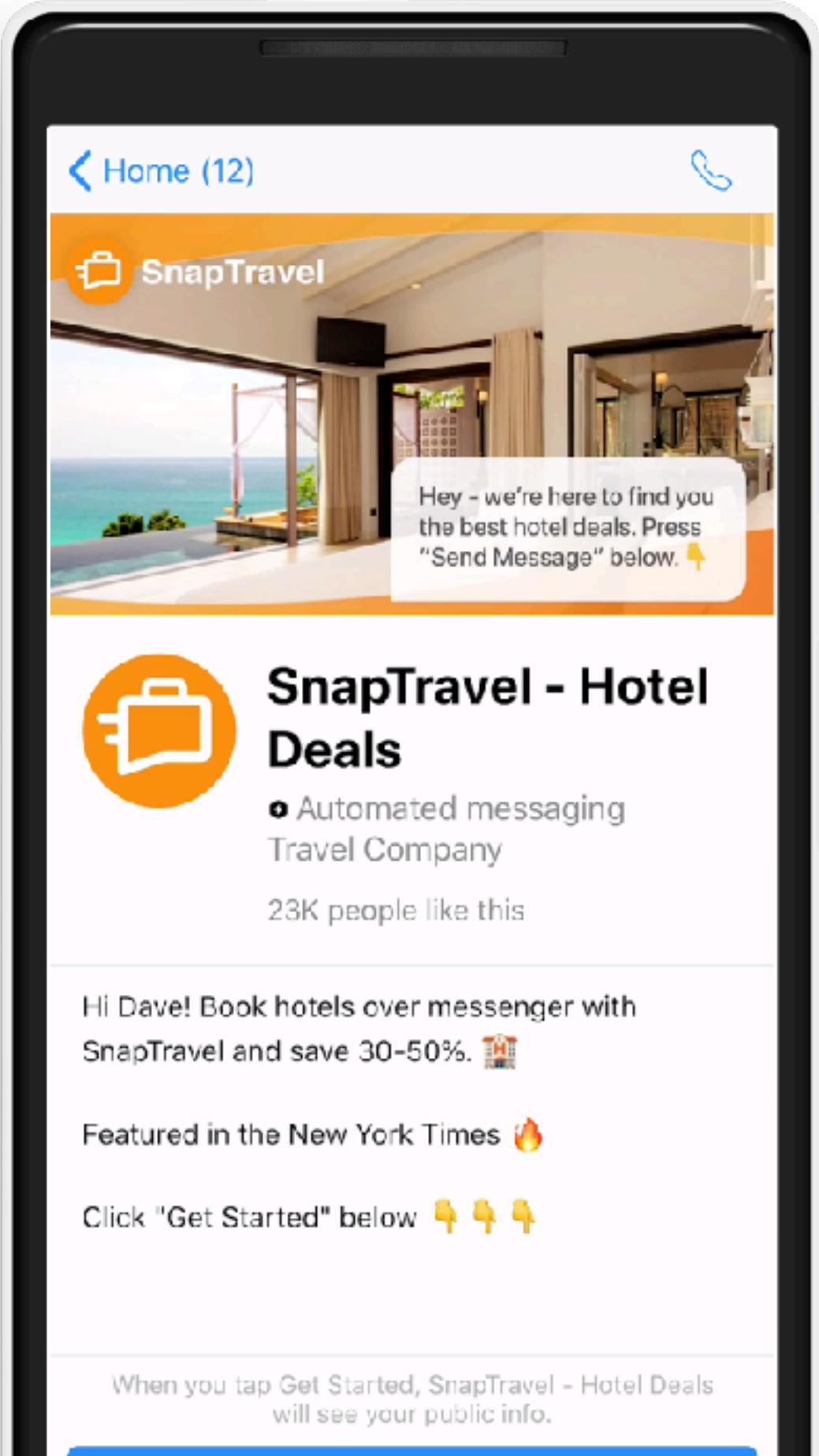
Riding bikes
in Santo Domingo

Biking near
a red building

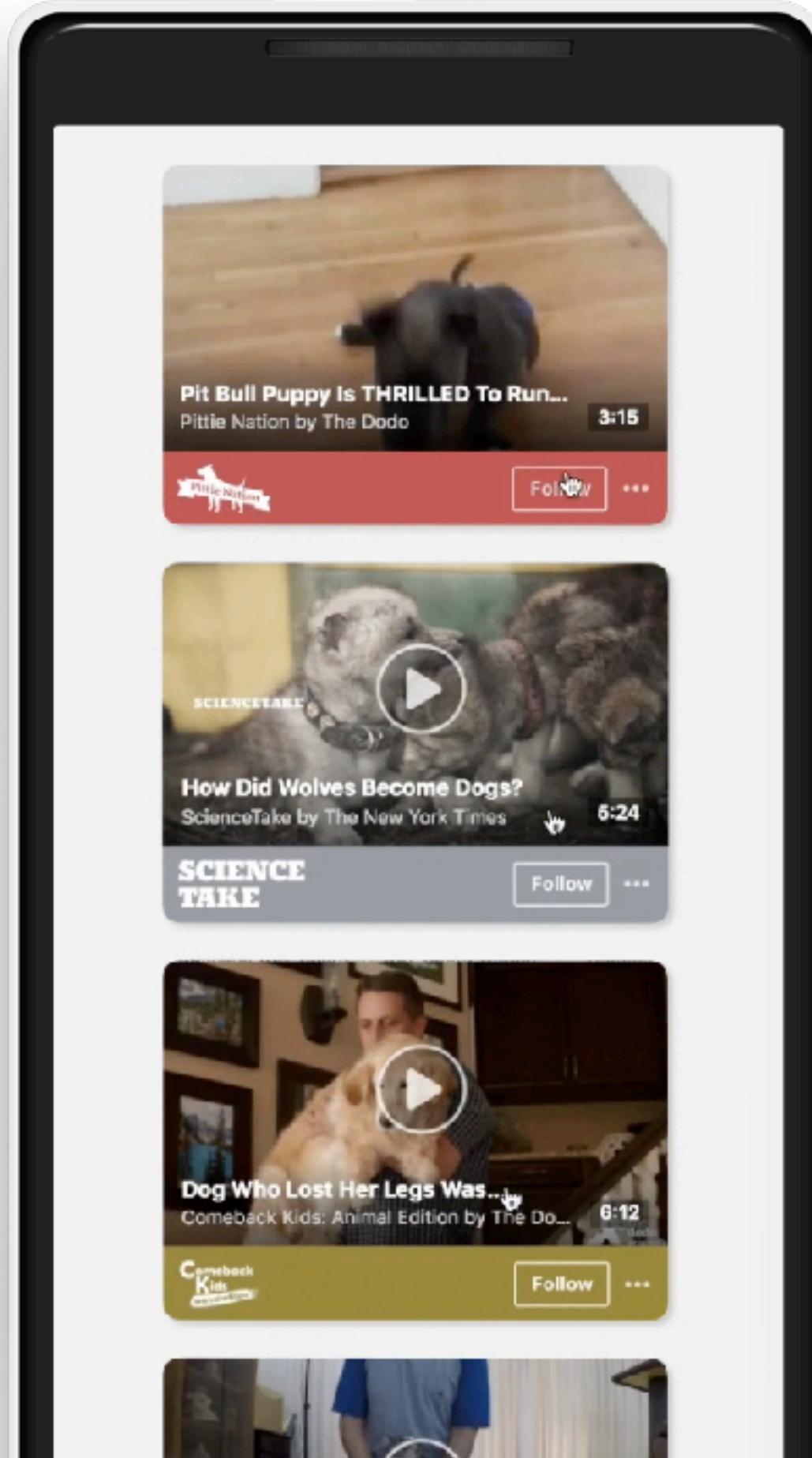


POWERING NEW EXPERIENCES

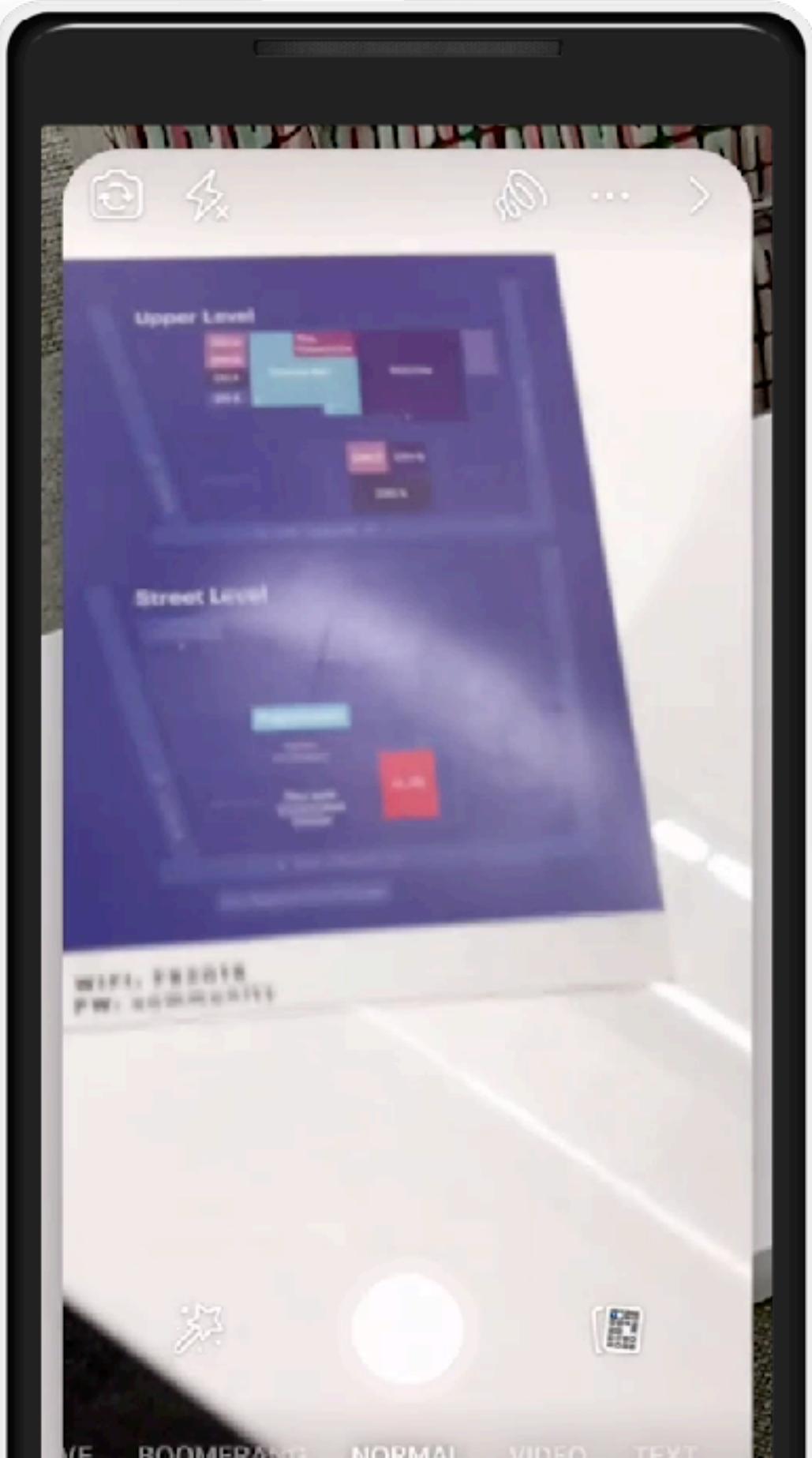
BOTS &
ASSISTANTS



GENERATED
CONTENT



AR
EFFECTS



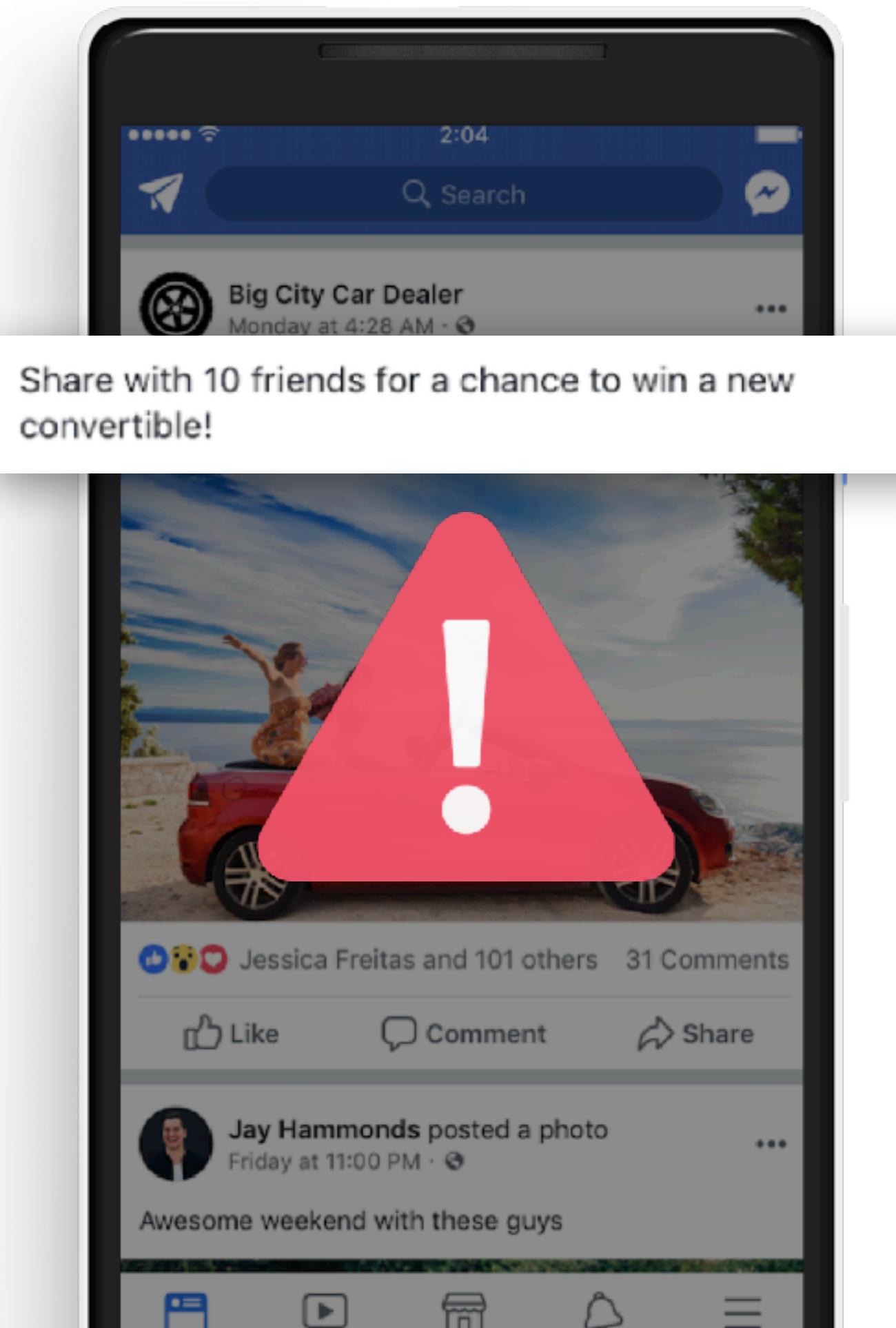
VR
HARDWARE





HELPING PROTECT COMMUNITY

SHARE BAITING

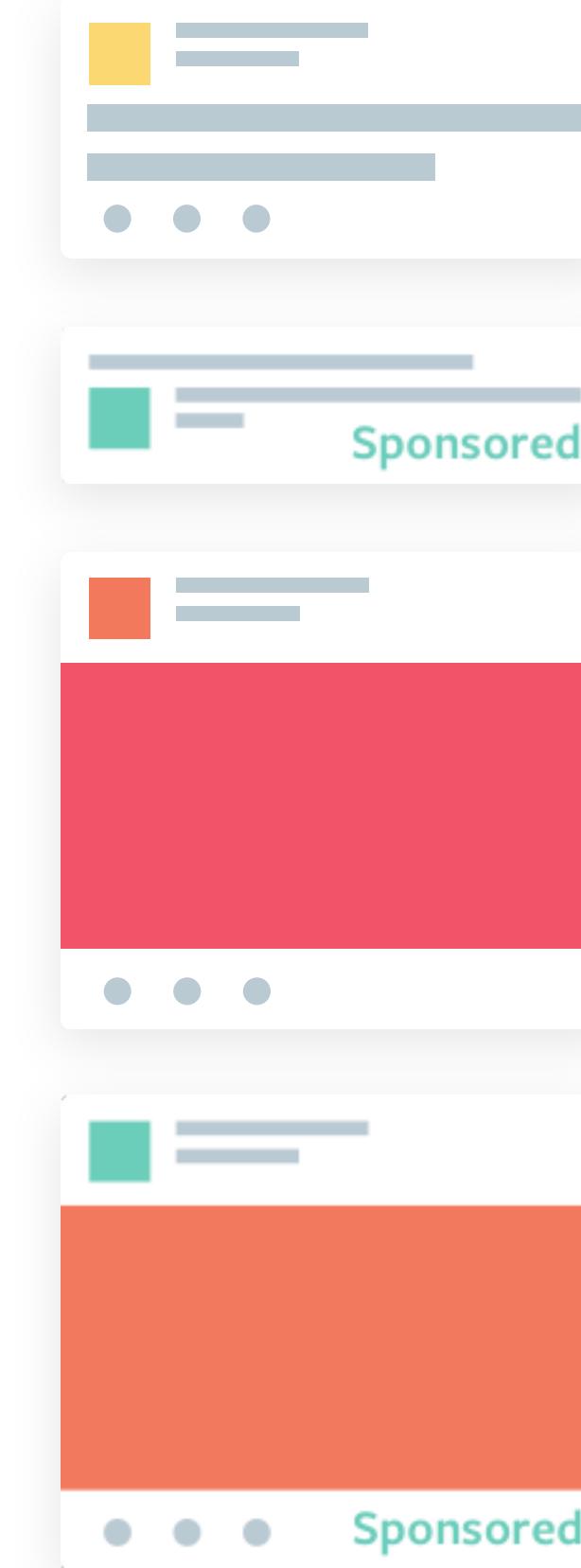
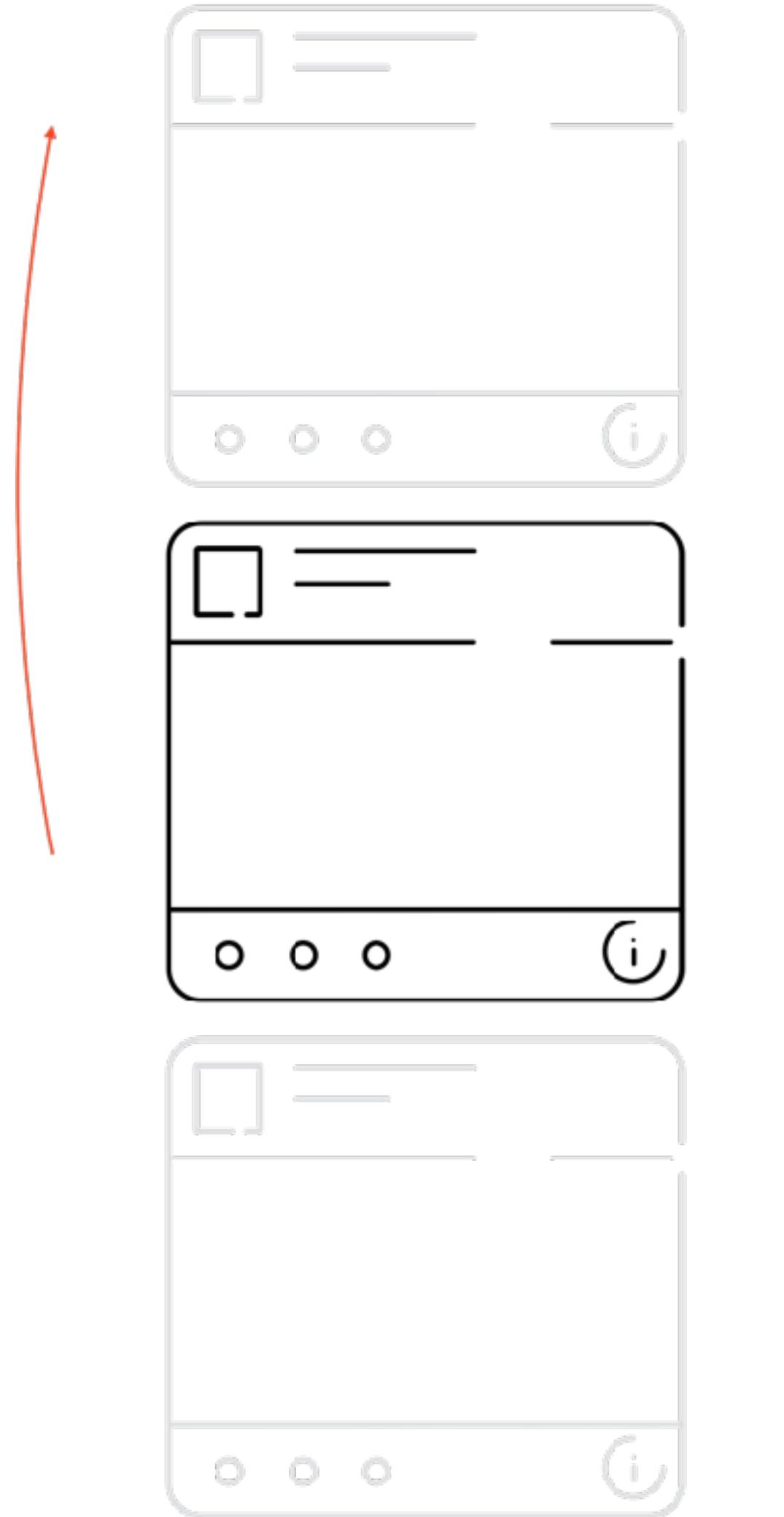
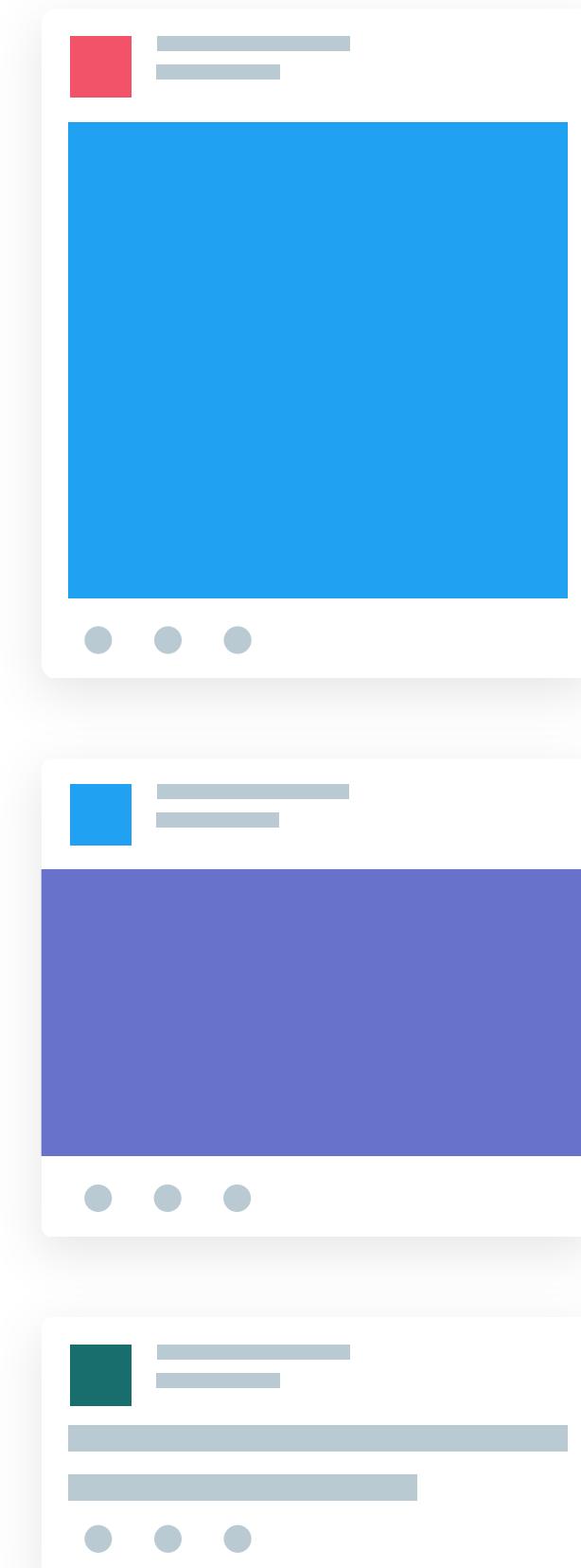
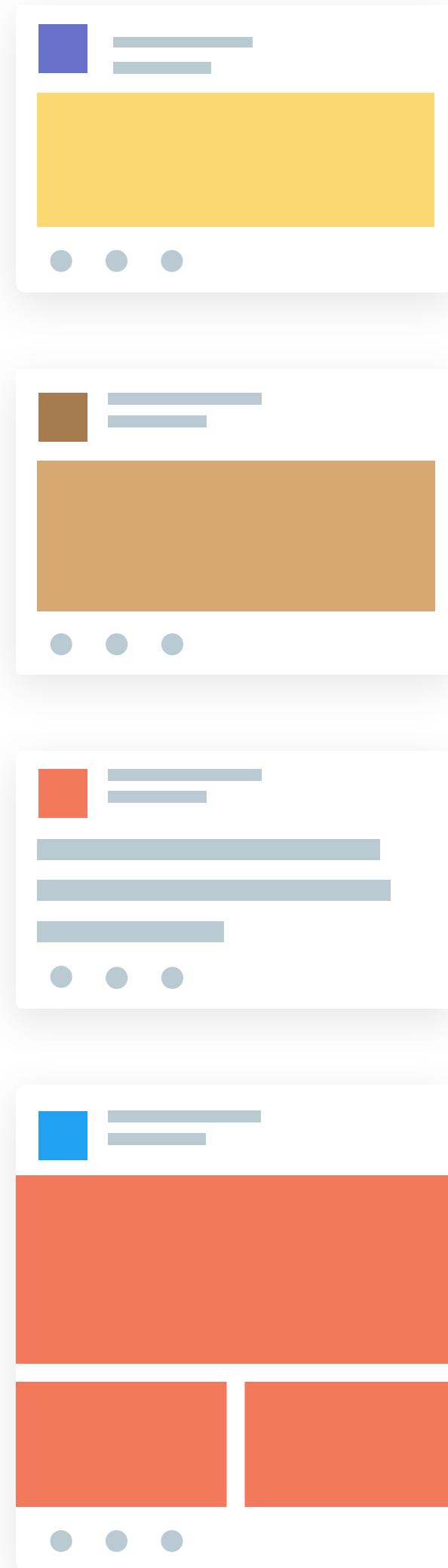


SUICIDE PREVENTION





NEWS FEED & ADS CUSTOMIZATION





Machine Learning in Facebook Products

SUPPORT VECTOR MACHINES

SVM

Facer

GRADIENT-BOOSTED DECISION TREES

GBDT

Integrity

MULTI-LAYER PERCEPTRON

MLP

News Feed

ADS

SEARCH

Integrity

CONVOLUTIONAL NEURAL NETS

CNN

Facer

Lumos

RECURRENT NEURAL NETS

RNN

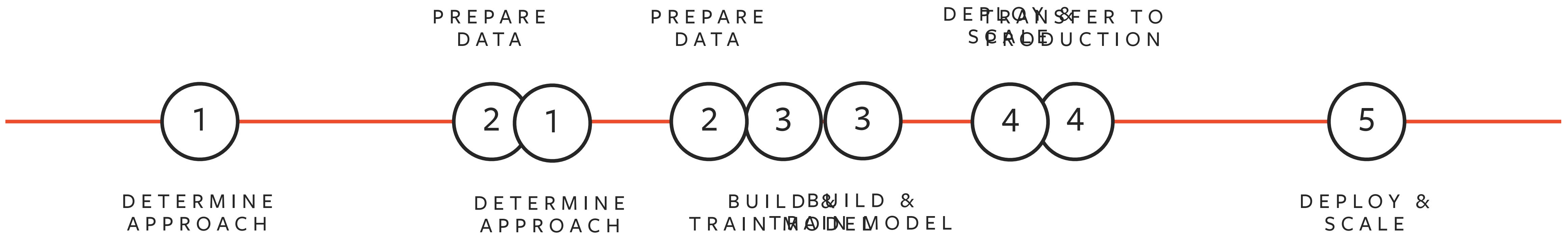
Language Translation

Speech Recognition

Content Understanding



Research to Production





Research to Production at Facebook – Early 2017

REIMPLEMENTATION TAKES WEEKS OR MONTHS

PYTORCH



Caffe2



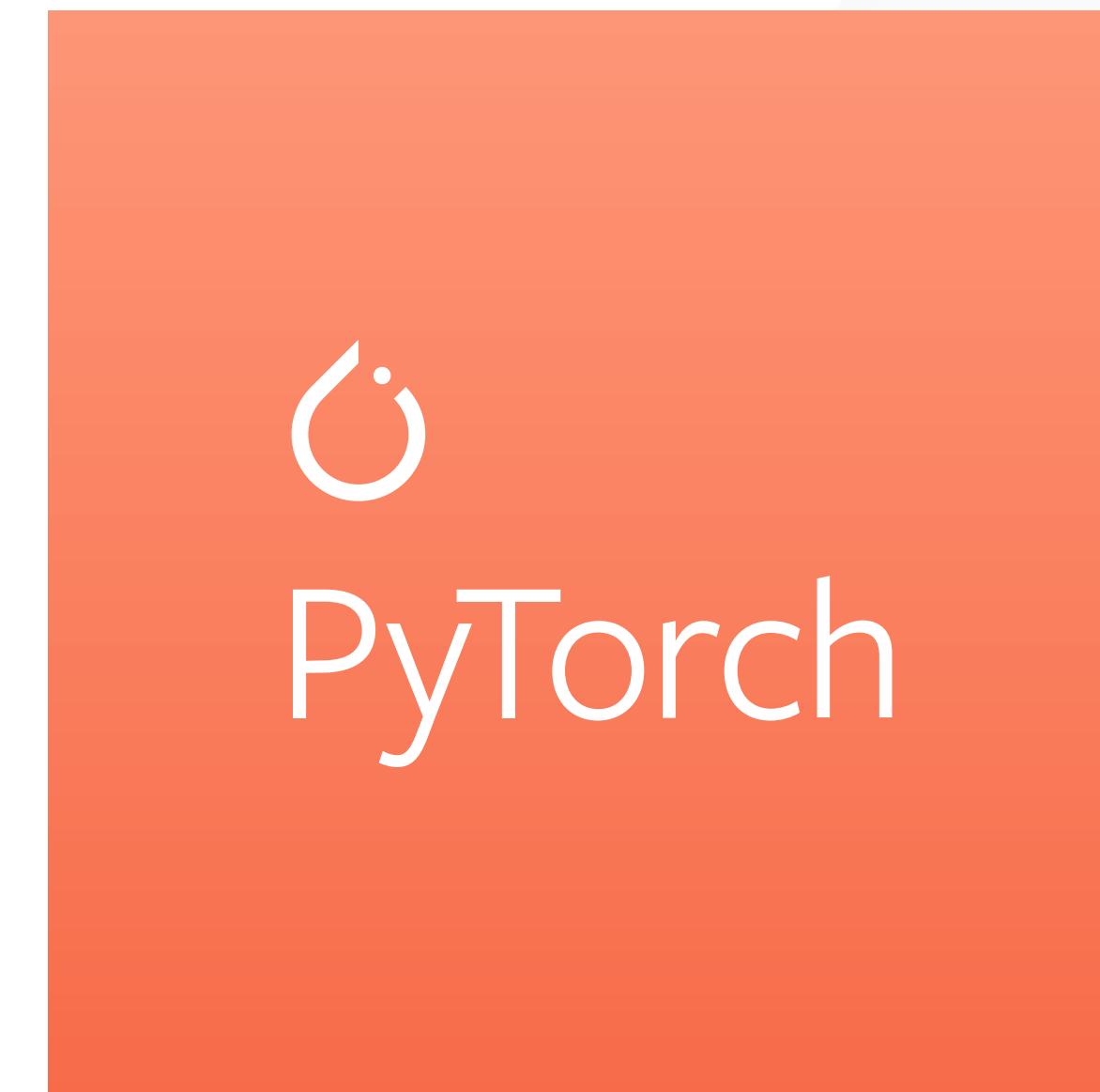
Research to Production at Facebook – Early 2018

ENABLING MODEL OR MODEL FRAGMENT TRANSFER





Research to Production at Facebook





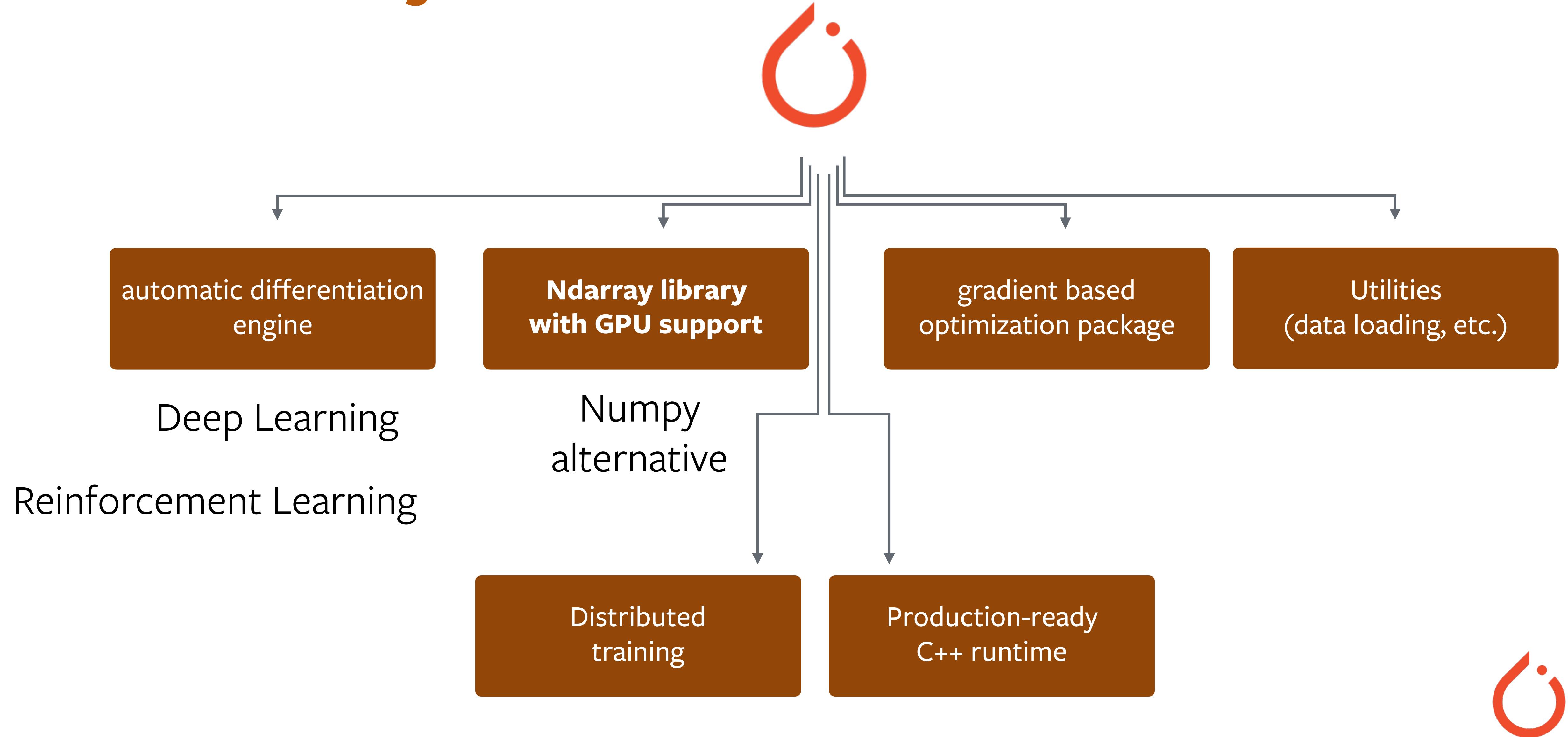
CORE TENETS

1. User-Centric
2. Community-Driven
3. Interoperable & Modular



What is PyTorch?

What is PyTorch?



ndarray library

- np.ndarray <-> torch.Tensor
- 200+ operations, similar to numpy
- very fast acceleration on NVIDIA GPUs



```

# -*- coding: utf-8 -*-
import numpy as np

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = np.random.randn(N, D_in)
y = np.random.randn(N, D_out)

# Randomly initialize weights
w1 = np.random.randn(D_in, H)
w2 = np.random.randn(H, D_out)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.dot(w1)
    h_relu = np.maximum(h, 0)
    y_pred = h_relu.dot(w2)

    # Compute and print loss
    loss = np.square(y_pred - y).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.T.dot(grad_y_pred)
    grad_h_relu = grad_y_pred.dot(w2.T)
    grad_h = grad_h_relu.copy()
    grad_h[h < 0] = 0
    grad_w1 = x.T.dot(grad_h)

    # Update weights
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

Numpy

```

import torch
dtype = torch.FloatTensor
# dtype = torch.cuda.FloatTensor # Uncomment this to run on GPU

# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random input and output data
x = torch.randn(N, D_in).type(dtype)
y = torch.randn(N, D_out).type(dtype)

# Randomly initialize weights
w1 = torch.randn(D_in, H).type(dtype)
w2 = torch.randn(H, D_out).type(dtype)

learning_rate = 1e-6
for t in range(500):
    # Forward pass: compute predicted y
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)

    # Compute and print loss
    loss = (y_pred - y).pow(2).sum()
    print(t, loss)

    # Backprop to compute gradients of w1 and w2 with respect to loss
    grad_y_pred = 2.0 * (y_pred - y)
    grad_w2 = h_relu.t().mm(grad_y_pred)
    grad_h_relu = grad_y_pred.mm(w2.t())
    grad_h = grad_h_relu.clone()
    grad_h[h < 0] = 0
    grad_w1 = x.t().mm(grad_h)

    # Update weights using gradient descent
    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2

```

PyTorch

ndarray / Tensor library

Tensors are similar to numpy's ndarrays, with the addition being that Tensors can also be used on a GPU to accelerate computing.

```
from __future__ import print_function  
import torch
```

Construct a 5x3 matrix, uninitialized:

```
x = torch.Tensor(5, 3)  
print(x)
```

Out:

```
1.00000e-25 *  
 0.4136  0.0000  0.0000  
 0.0000  1.6519  0.0000  
 1.6518  0.0000  1.6519  
 0.0000  1.6518  0.0000  
 1.6520  0.0000  1.6519  
[torch.FloatTensor of size 5x3]
```



ndarray / Tensor library

Construct a randomly initialized matrix

```
x = torch.rand(5, 3)
print(x)
```

Out:

```
0.2598  0.7231  0.8534
0.3928  0.1244  0.5110
0.5476  0.2700  0.5856
0.7288  0.9455  0.8749
0.6663  0.8230  0.2713
[torch.FloatTensor of size 5x3]
```

Get its size

```
print(x.size())
```

Out:

```
torch.Size([5, 3])
```



ndarray / Tensor library

You can use standard numpy-like indexing with all bells and whistles!

```
print(x[:, 1])
```

Out:

```
0.7231
0.1244
0.2700
0.9455
0.8230
[torch.FloatTensor of size 5]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

Out:

```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



NumPy bridge

Converting torch Tensor to numpy Array

```
a = torch.ones(5)  
print(a)
```

Out:

```
1  
1  
1  
1  
1  
[torch.FloatTensor of size 5]
```

**Zero memory-copy
very efficient**

```
b = a.numpy()  
print(b)
```

Out:

```
[ 1.  1.  1.  1.  1.]
```



NumPy bridge

See how the numpy array changed in value.

```
a.add_(1)  
print(a)  
print(b)
```

Out:

```
2  
2  
2  
2  
2  
[torch.FloatTensor of size 5]  
[ 2.  2.  2.  2.  2.]
```



NumPy bridge

Converting numpy Array to torch Tensor

See how changing the np array changed the torch Tensor automatically

```
import numpy as np
a = np.ones(5)
b = torch.from_numpy(a)
np.add(a, 1, out=a)
print(a)
print(b)
```

Out:

```
[ 2.  2.  2.  2.  2.]
2
2
2
2
2
[torch.DoubleTensor of size 5]
```

All the Tensors on the CPU except a CharTensor support converting to NumPy and back.



Seamless GPU Tensors

CUDA Tensors

Tensors can be moved onto GPU using the `.cuda` function.

```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    x = x.cuda()
    y = y.cuda()
    x + y
```

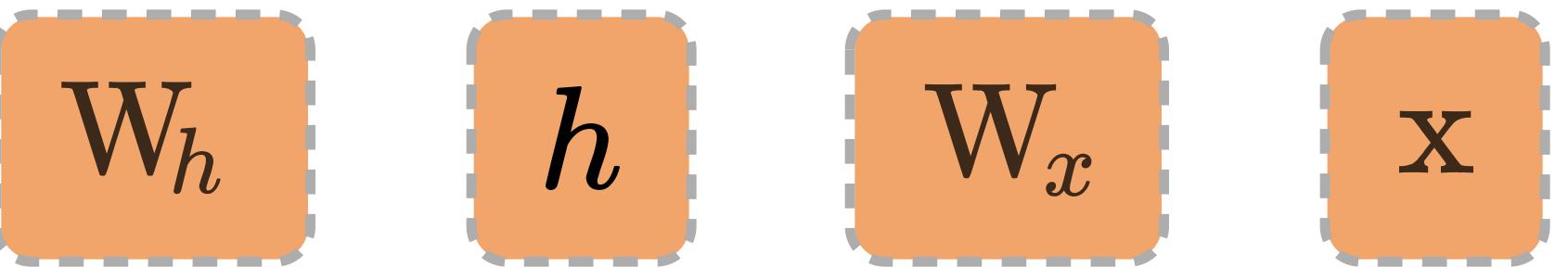


automatic differentiation engine

for deep learning and reinforcement learning



PyTorch Autograd

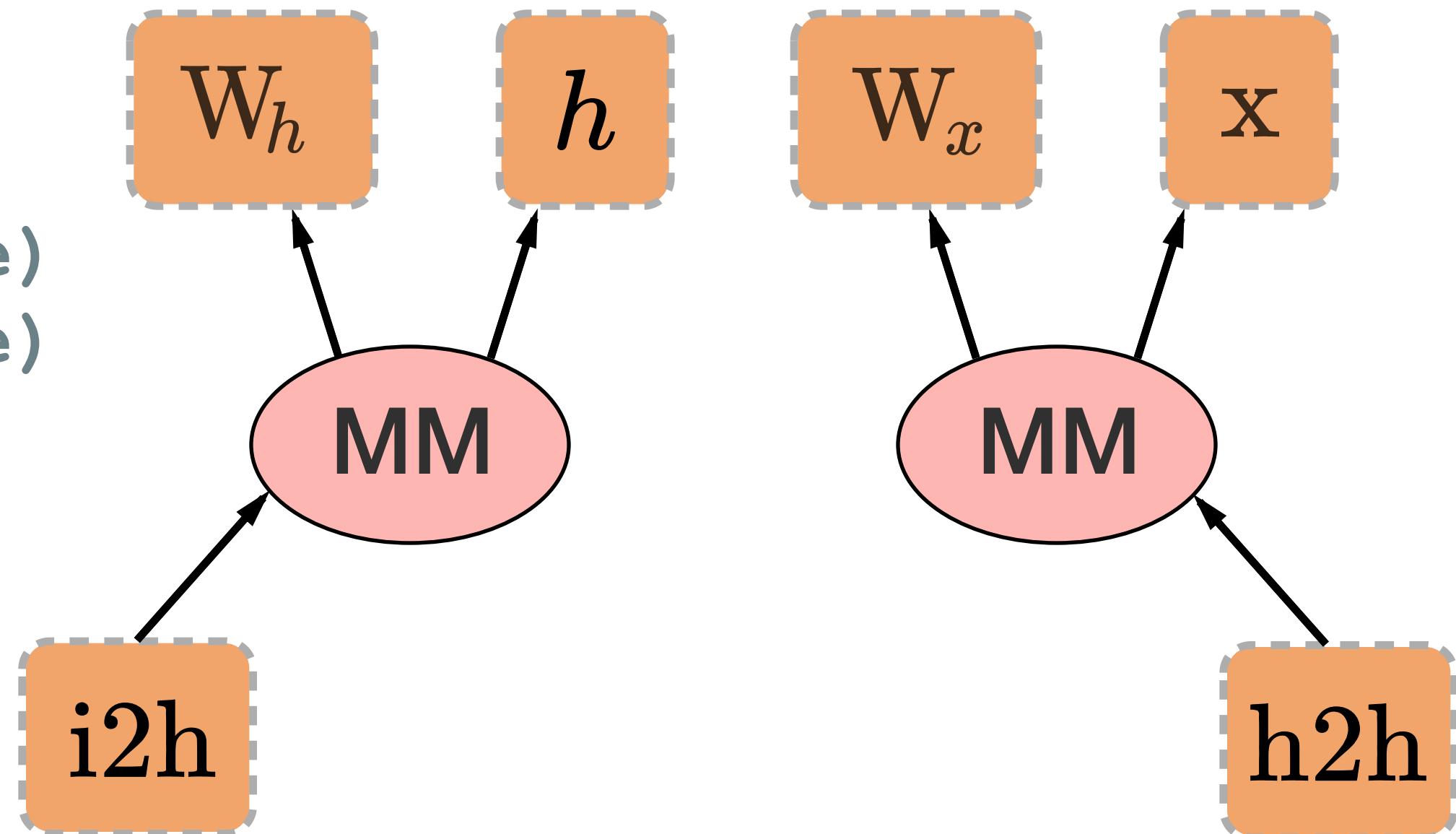


```
w_h = torch.randn(20, 20, requires_grad=True)
w_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

PyTorch Autograd

```
w_h = torch.randn(20, 20, requires_grad=True)
w_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

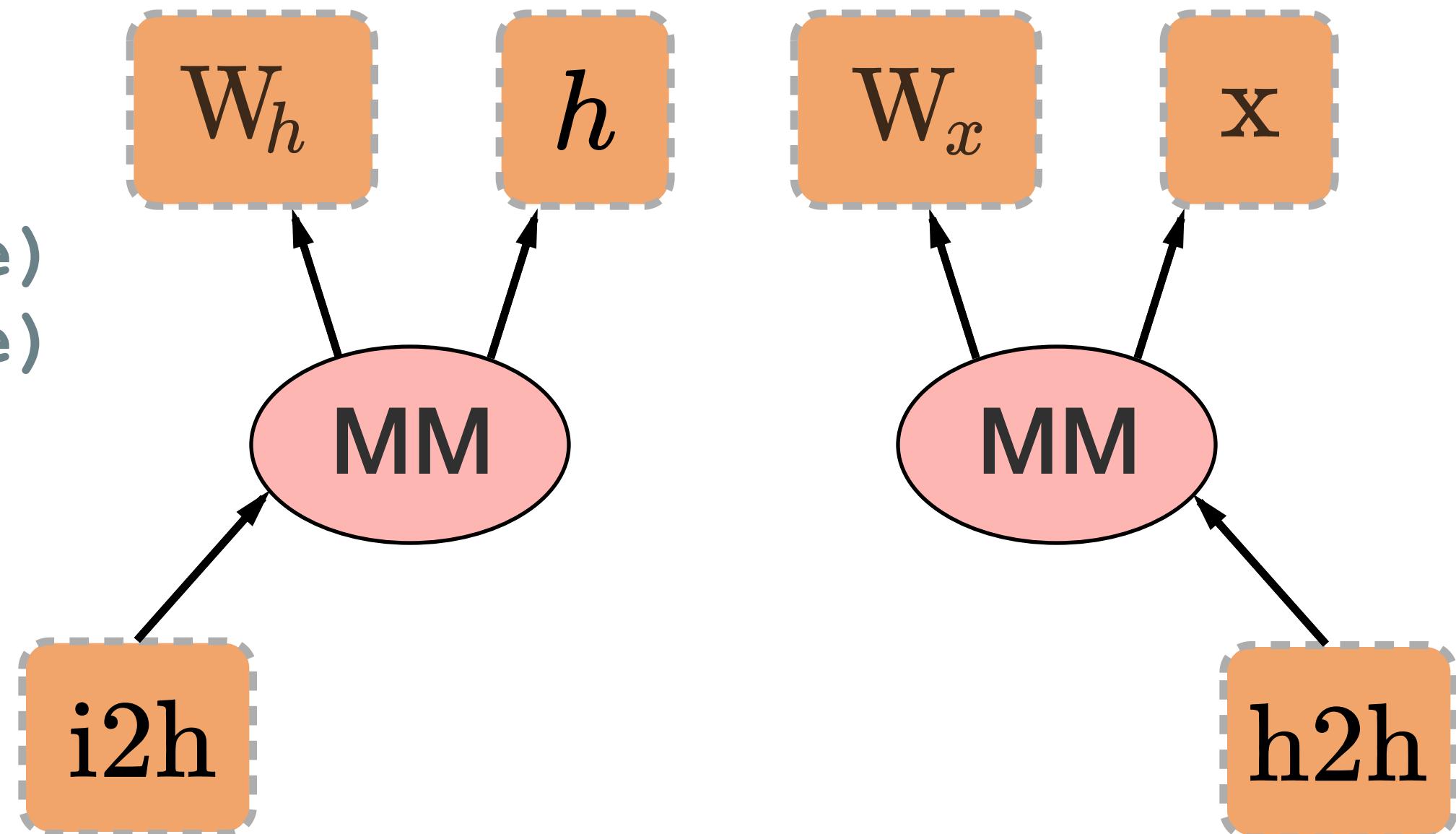
```
i2h = torch.mm(w_x, x.t())
h2h = torch.mm(w_h, prev_h.t())
```



PyTorch Autograd

```
w_h = torch.randn(20, 20, requires_grad=True)
w_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

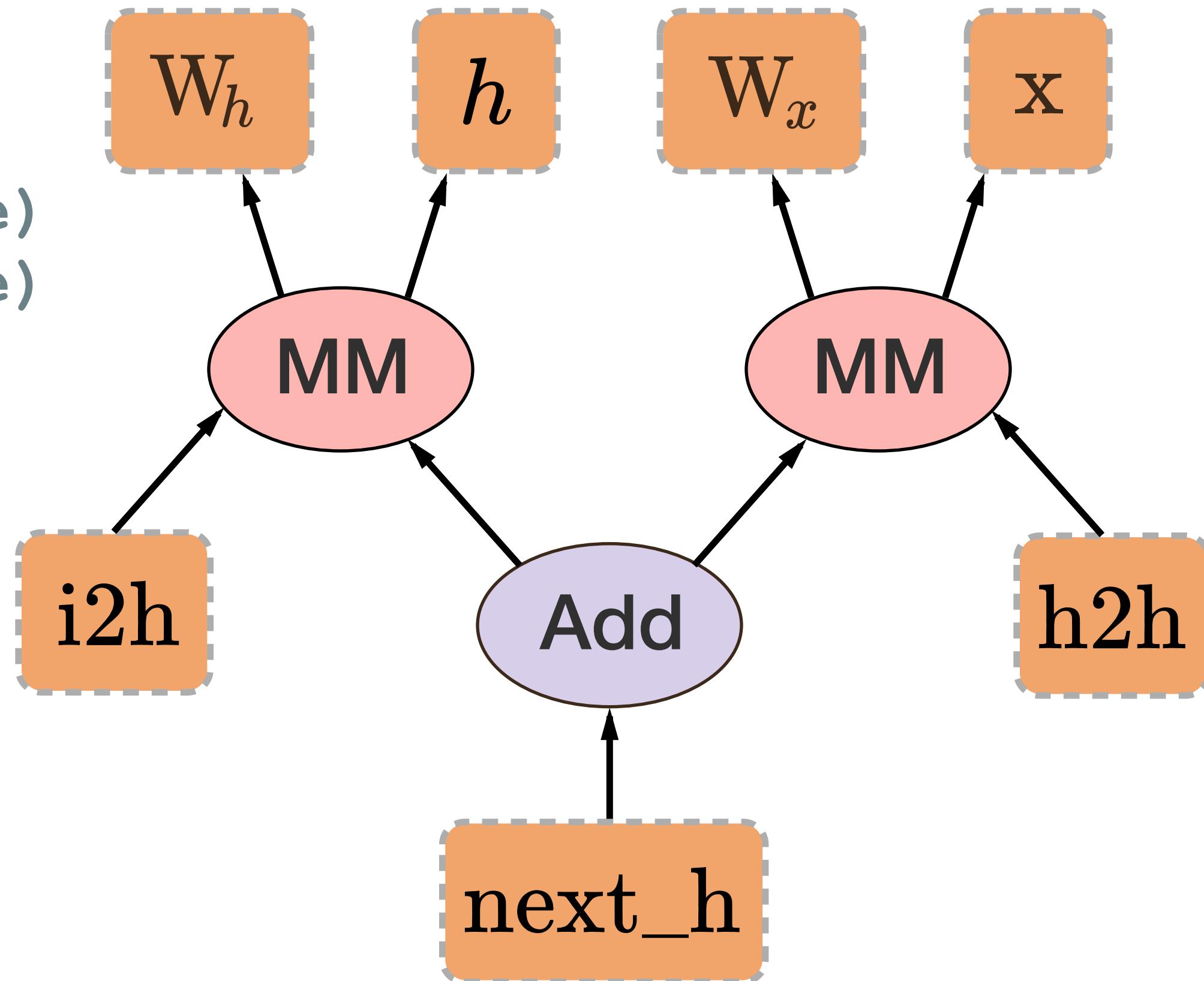
```
i2h = torch.mm(w_x, x.t())
h2h = torch.mm(w_h, prev_h.t())
next_h = i2h + h2h
```



PyTorch Autograd

```
w_h = torch.randn(20, 20, requires_grad=True)
w_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

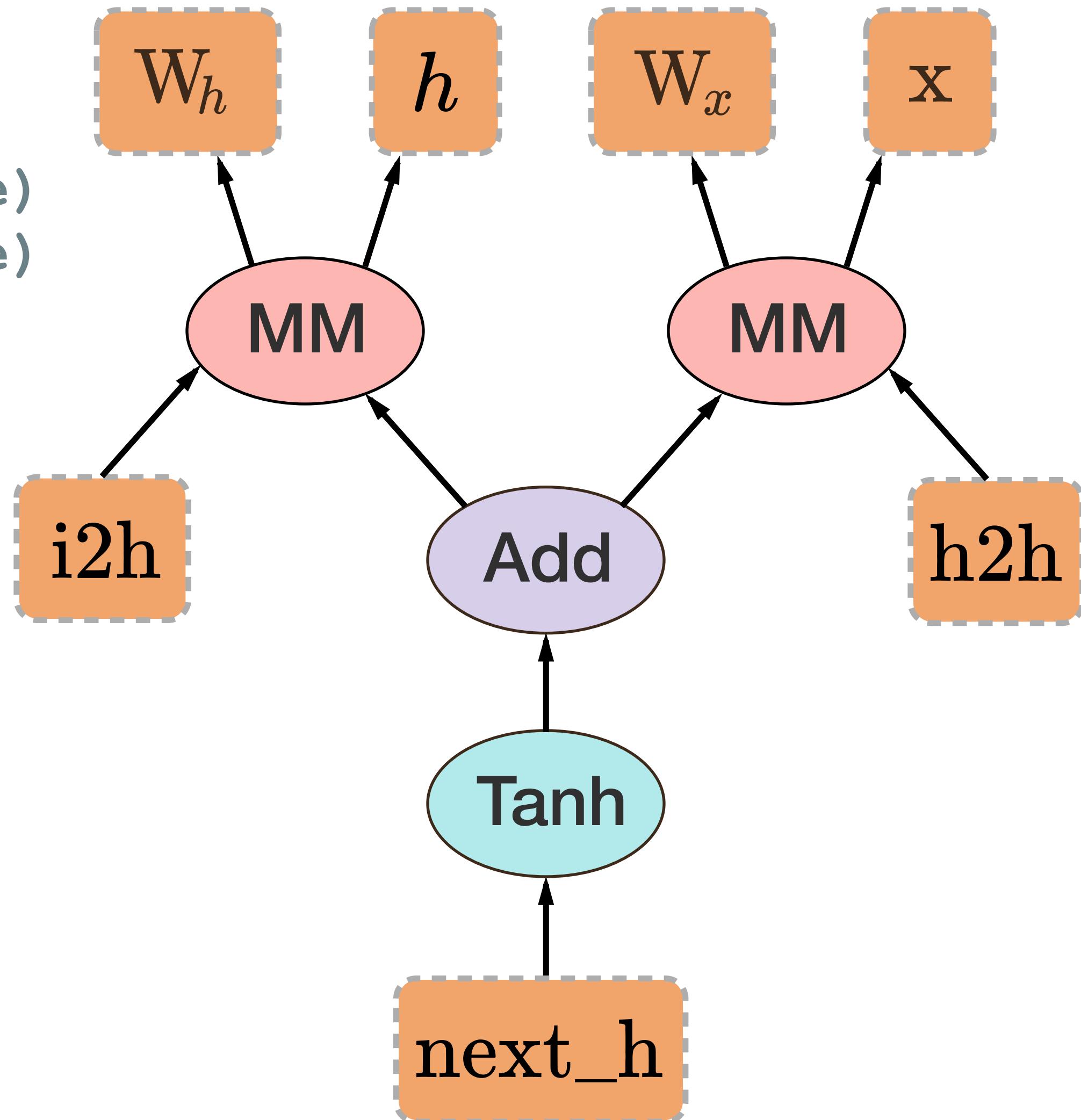
```
i2h = torch.mm(w_x, x.t())
h2h = torch.mm(w_h, prev_h.t())
next_h = i2h + h2h
```



PyTorch Autograd

```
w_h = torch.randn(20, 20, requires_grad=True)
w_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

```
i2h = torch.mm(w_x, x.t())
h2h = torch.mm(w_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()
```



PyTorch Autograd

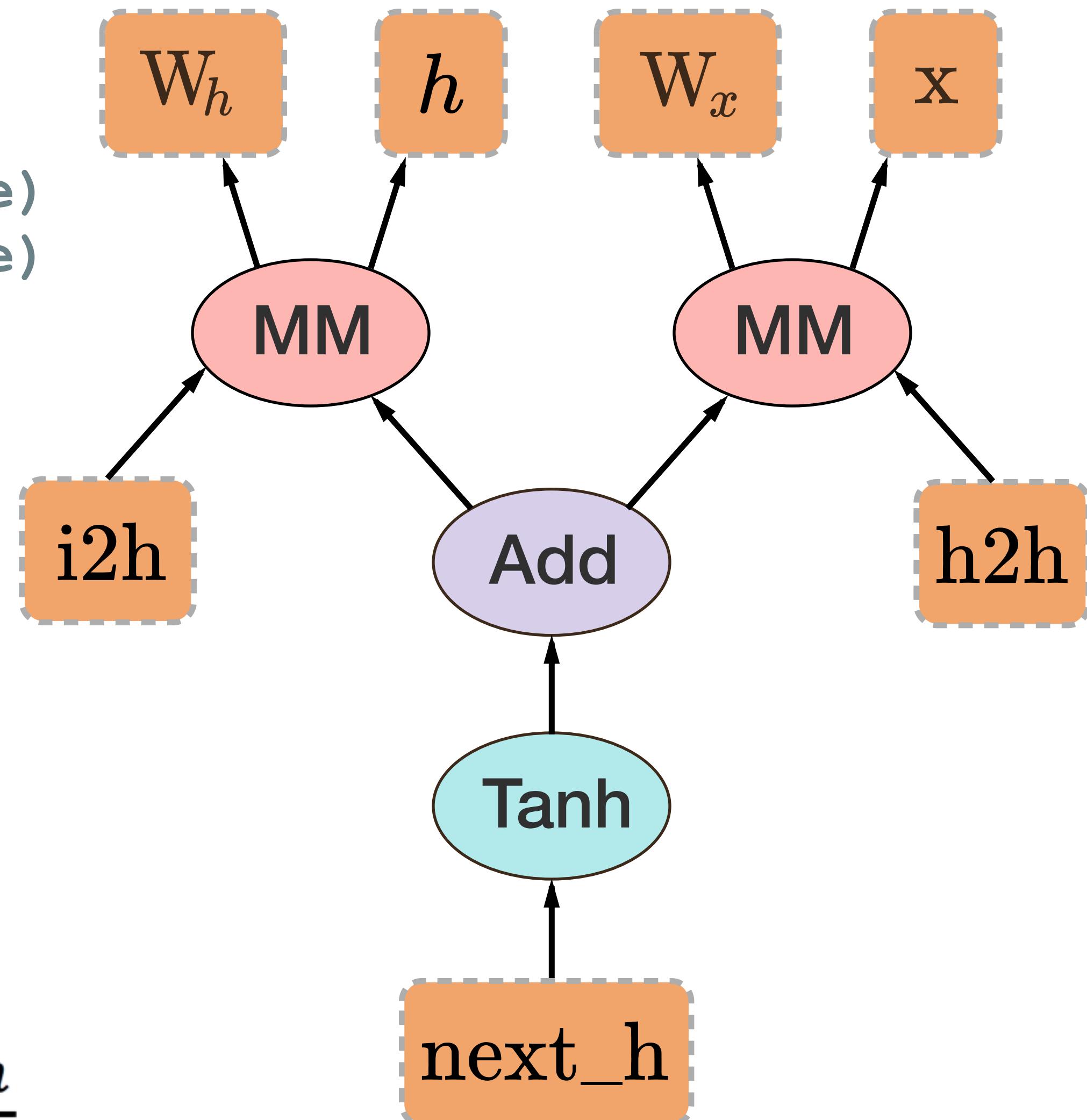
```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```

```
i2h = torch.mm(W_x, x.t())
h2h = torch.mm(W_h, prev_h.t())
next_h = i2h + h2h
next_h = next_h.tanh()
```

```
next_h.backward(torch.ones(1, 20))
```

$$\frac{d\text{next}_h}{dW_h}$$

$$W_h = W_h - \alpha * \frac{d\text{next}_h}{dW_h}$$



Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Neural Networks

```
1  class Net(nn.Module):
2      def __init__(self):
3          super(Net, self).__init__()
4          self.conv1 = nn.Conv2d(1, 10, kernel_size=5)
5          self.conv2 = nn.Conv2d(10, 20, kernel_size=5)
6          self.conv2_drop = nn.Dropout2d()
7          self.fc1 = nn.Linear(320, 50)
8          self.fc2 = nn.Linear(50, 10)
9
10     def forward(self, x):
11         x = F.relu(F.max_pool2d(self.conv1(x), 2))
12         x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
13         x = x.view(-1, 320)
14         x = F.relu(self.fc1(x))
15         x = F.dropout(x, training=self.training)
16         x = self.fc2(x)
17         return F.log_softmax(x)
18
19 model = Net()
20 input = Variable(torch.randn(10, 20))
21 output = model(input)
```

Optimization package

SGD, Adagrad, RMSProp, LBFGS, etc.

```
1 net = Net()
2 optimizer = torch.optim.SGD(net.parameters(), lr=0.01, momentum=0.9)
3
4 for input, target in dataset:
5     optimizer.zero_grad()
6     output = model(input)
7     loss = F.cross_entropy(output, target)
8     loss.backward()
9     optimizer.step()
```

Debugging

- PyTorch is a Python extension



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger
- Use the most popular debugger:



Debugging

- PyTorch is a Python extension
- Use your favorite Python debugger
- Use the most popular debugger:

print (foo)

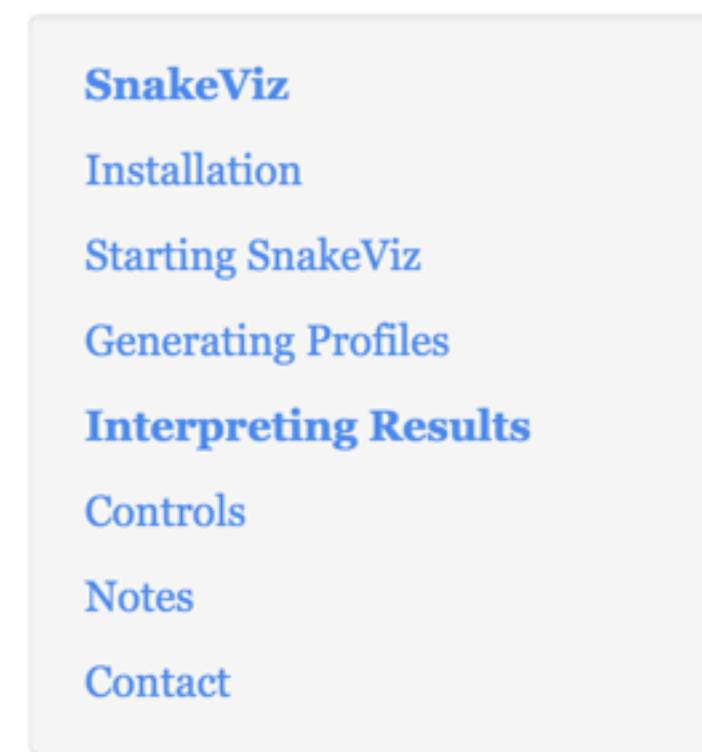


Identifying bottlenecks

- PyTorch is a Python extension
- Use your favorite Python profiler

SNAKEVIZ

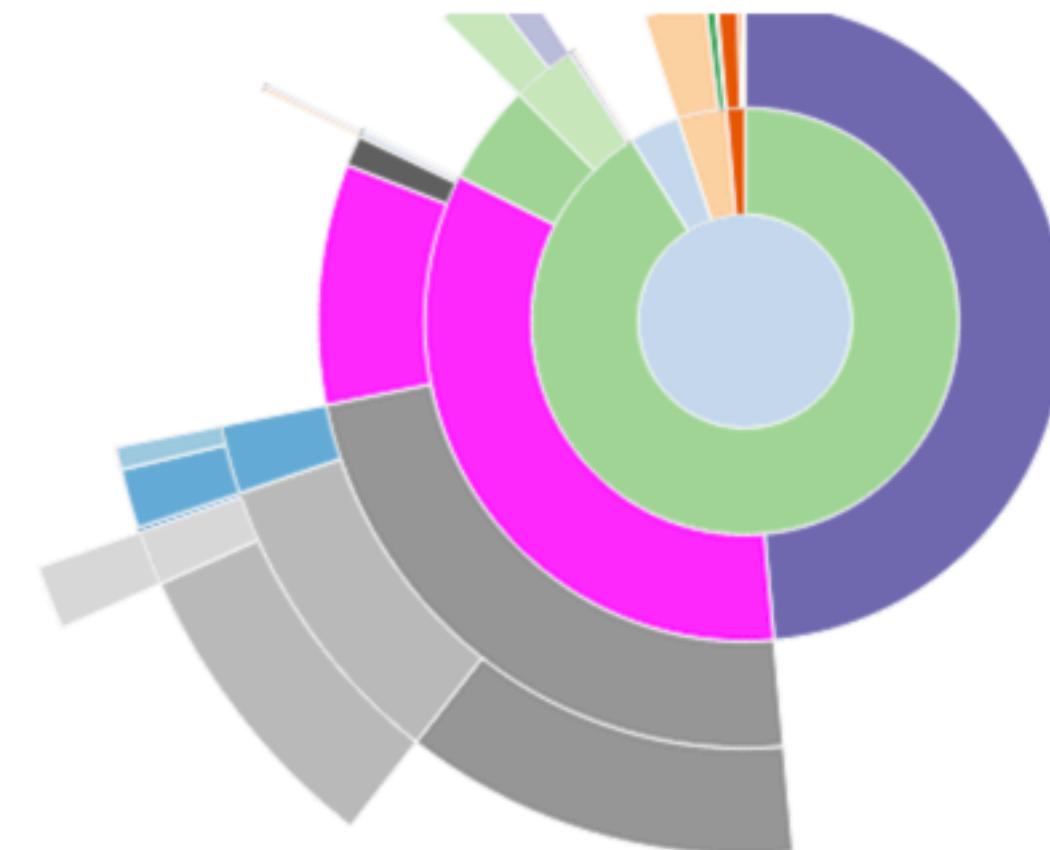
← PREVIOUS NEXT → OG



FUNCTION INFO

Placing your cursor over an arc will highlight that arc and any other visible instances of the same function call. It also display a list of information to the left of the sunburst.

Name:
filter
Cumulative Time:
0.000294 s (31.78 %)
File:
fnmatch.py
Line:
48
Directory:
/Users/jiffyclub/miniconda3/en
vs/snakevizdev/lib/python3.4/



Identifying bottlenecks

- PyTorch is a Python extension
- Use your favorite Python profiler: Line_Profiler

```
File: pystone.py
Function: Proc2 at line 149
Total time: 0.606656 s

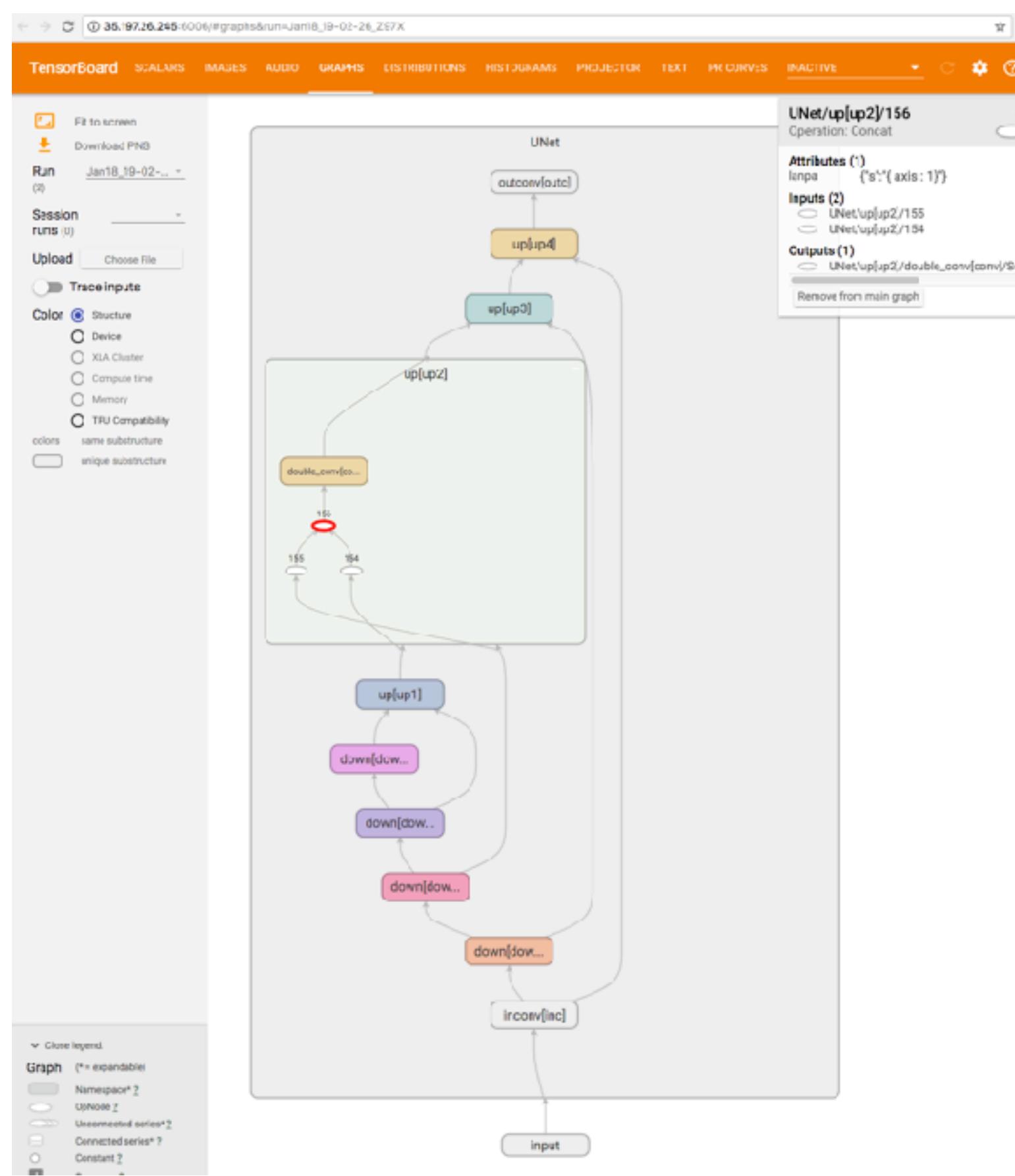
Line #      Hits       Time  Per Hit   % Time  Line Contents
=====
149                      @profile
150                      def Proc2(IntParIO):
151    50000     82003     1.6    13.5
152    50000     63162     1.3    10.4
153    50000     69065     1.4    11.4
154    50000     66354     1.3    10.9
155    50000     67263     1.3    11.1
156    50000     65494     1.3    10.8
157    50000     68001     1.4    11.2
158    50000     63739     1.3    10.5
159    50000     61575     1.2    10.1
                                         IntLoc = IntParIO + 10
                                         while 1:
                                         if Char1Glob == 'A':
                                         IntLoc = IntLoc - 1
                                         IntParIO = IntLoc - IntGlob
                                         EnumLoc = Ident1
                                         if EnumLoc == Ident1:
                                         break
                                         return IntParIO
```



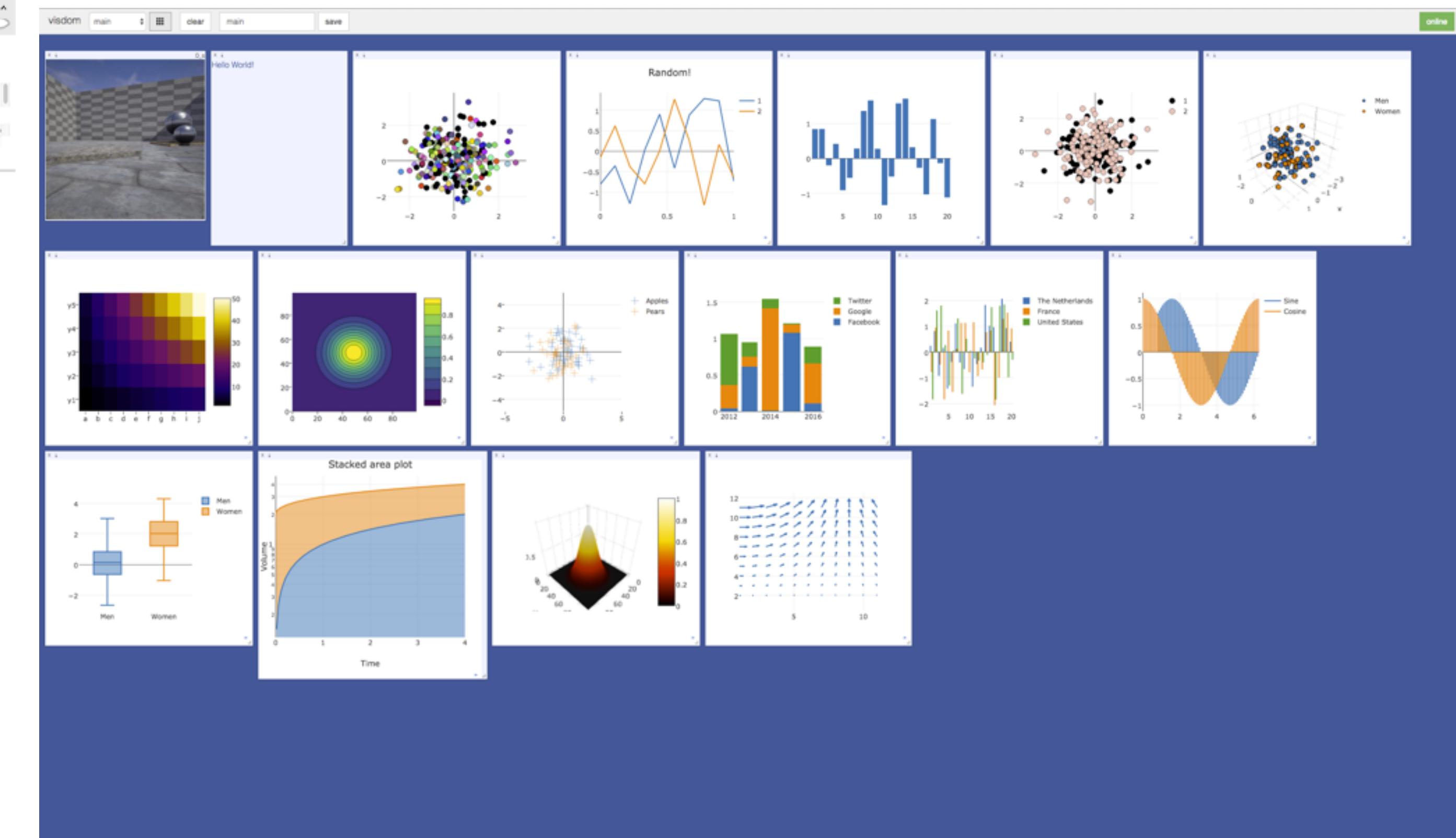
Visualization

TensorBoard-PyTorch

<https://github.com/lanpa/tensorboard-pytorch>



<https://github.com/facebookresearch/visdom>



Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Python + PyTorch - an environment to do all of this

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines



Work items in practice

Writing
Dataset loaders

Building models

Implementing
Training loop

Checkpointing
models

Python + PyTorch - an environment to do all of this

Interfacing with
environments

Building optimizers

Dealing with
GPUs

Building
Baselines

+ tools for making your model production-ready

Distributed
training

Performance
optimization

Deployment
for inference





Imperative vs Declarative Toolkits



DEVELOPER EFFICIENCY

Debuggability

Interactivity

Simplicity

Intuitiveness

INFRASTRUCTURE EFFICIENCY

Efficiency

Reliability

Scalability

Cross Platform



COMPUTATION GRAPH

Declarative
Toolkits

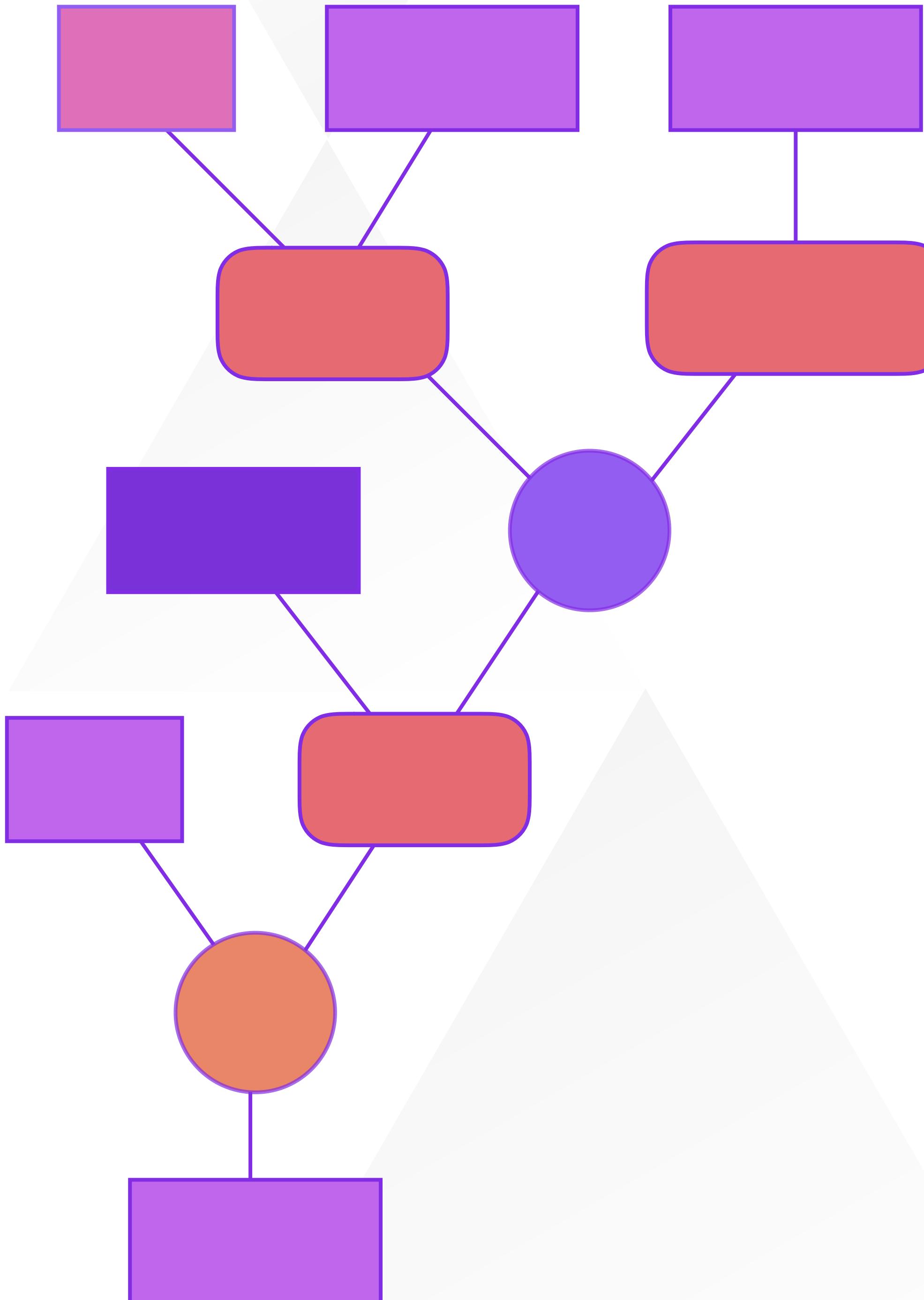
theano

Caffe

mxnet

 Caffe2

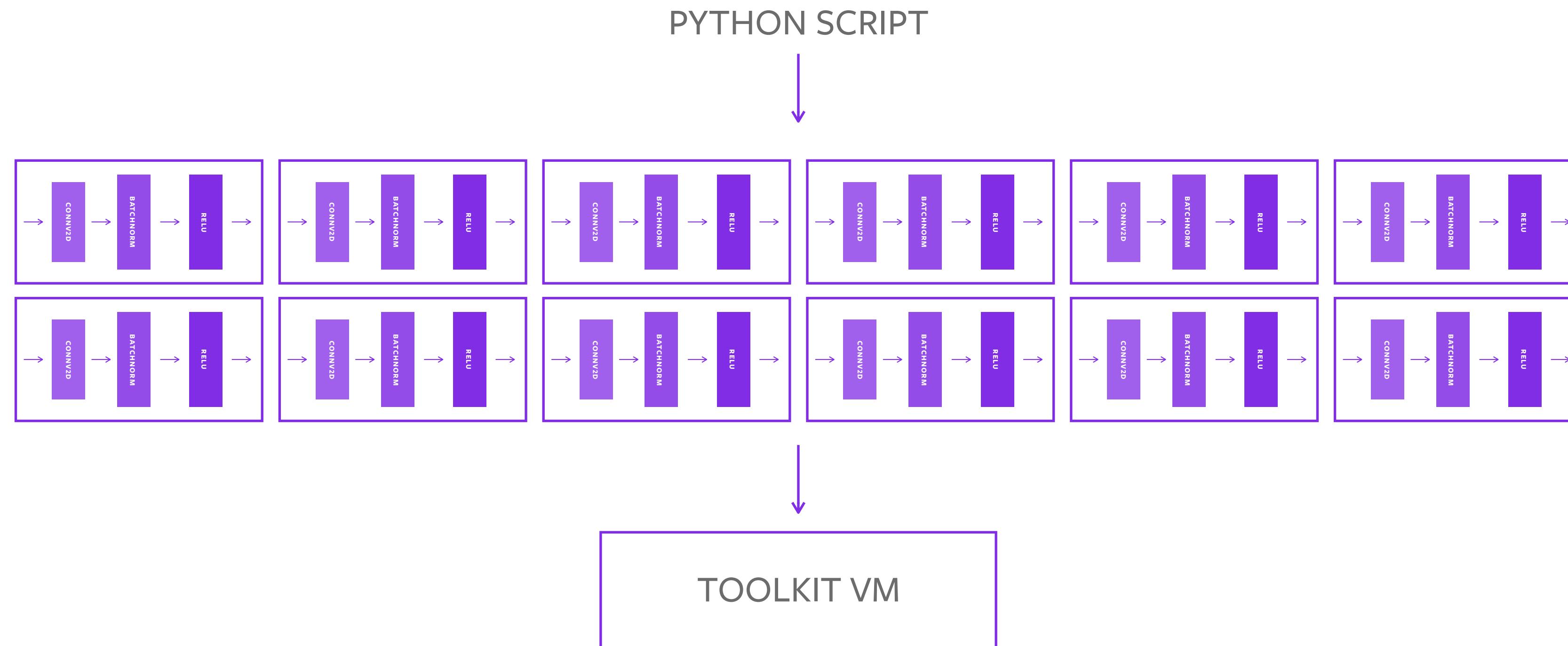
 TensorFlow

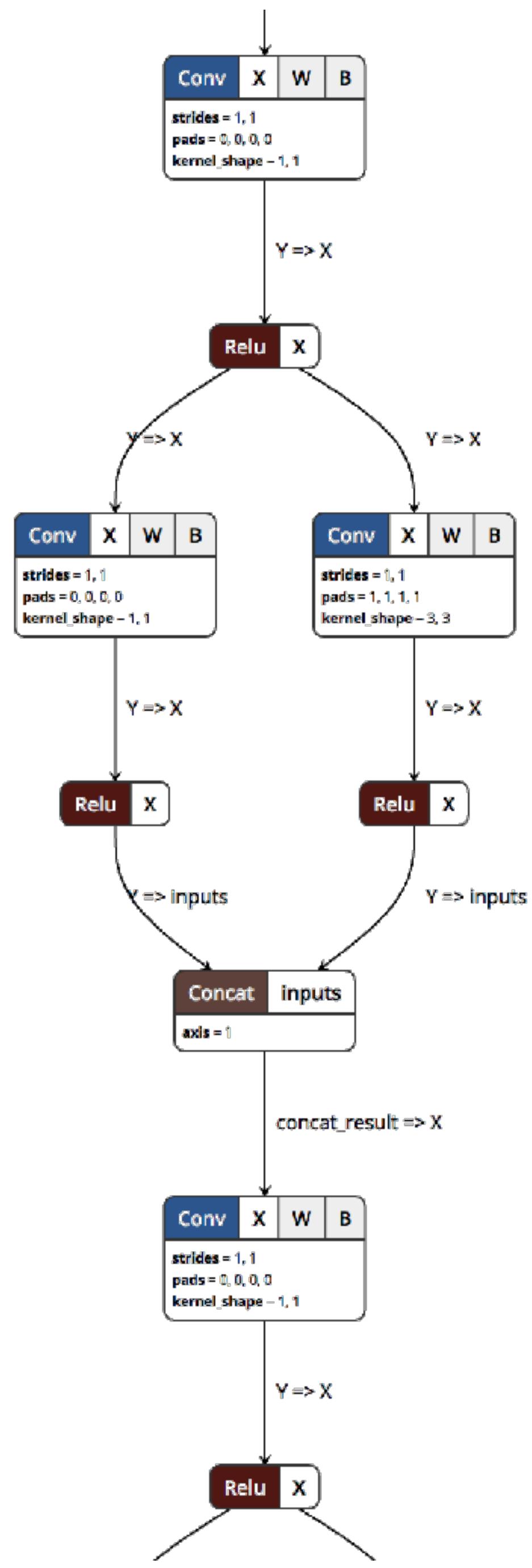




DECLARATIVE TOOLKITS

Declare and compile a model
Repeatedly execute the model in a VM





DOCUMENTATION

X

Relu

Relu takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, $y = \max(0, x)$, is applied to the tensor elementwise.

Attributes

consumed_inputs: int[]

legacy optimization attribute.

Inputs

X: T

Input tensor

Outputs

Y: T

Output tensor

Type Constraints

T: tensor(float16), tensor(float), tensor(double)

Constrain input and output types to float tensors.

Examples

relu

```
node = onnx.helper.make_node(
    'Relu',
    inputs=['x'],
    outputs=['y'],
)
x = np.random.randn(3, 4, 5).astype(np.float32)
y = np.clip(x, 0, np.inf)

expect(node, inputs=[x], outputs=[y],
       name='test_relu')
```

Support

In domain `ai.onnx` since version `1` at support level `common`.



Computation Graph

DECLARATIVE TOOLKITS

- Declare a computation
- Placeholder variables
- Compile it
- Run it in a Session

```
import tensorflow as tf
import numpy as np
trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
X = tf.placeholder("float")
Y = tf.placeholder("float")
def model(X, w):
    return tf.multiply(X, w)
w = tf.Variable(0.0, name="weights")
y_model = model(X, w)
cost = tf.square(Y - y_model)
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(100):
        for (x, y) in zip(trX, trY):
            sess.run(train_op, feed_dict={X: x, Y: y})
    print(sess.run(w))
```



Computation Graph

DECLARATIVE TOOLKITS

- Declare a computation
- Placeholder variables
- Compile it
- Run it in a Session

```
import tensorflow as tf
import numpy as np
trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33

X = tf.placeholder("float")
Y = tf.placeholder("float")

def model(X, w):
    return tf.multiply(X, w)

w = tf.Variable(0.0, name="weights")
y_model = model(X, w)
cost = tf.square(Y - y_model)
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

with tf.Session() as sess:
    tf.global_variables_initializer().run()
    for i in range(100):
        for (x, y) in zip(trX, trY):
            sess.run(train_op, feed_dict={X: x, Y: y})
    print(sess.run(w))
```



Computation Graph

DECLARATIVE TOOLKITS

- Declare a computation
- Placeholder variables
- Compile it
- Run it in a Session

```
import tensorflow as tf
import numpy as np
trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
X = tf.placeholder("float")
```

Model definition

```
def model(X, w):
    return tf.multiply(X, w)
w = tf.Variable(0.0, name="weights")
y_model = model(X, w)
cost = tf.square(Y - y_model)
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)
```

```
tf.global_variables_initializer().run()
for i in range(100):
    for (x, y) in zip(trX, trY):
        sess.run(train_op, feed_dict={X: x, Y: y})
    print(sess.run(w))
```



Computation Graph

DECLARATIVE TOOLKITS

- Declare a computation
- Placeholder variables
- Compile it
- Run it in a Session

```
import tensorflow as tf
import numpy as np
trX = np.linspace(-1, 1, 101)
trY = 2 * trX + np.random.randn(*trX.shape) * 0.33
X = tf.placeholder("float")
Y = tf.placeholder("float")
def model(X, w):
    return tf.multiply(X, w)
w = tf.Variable(0.0, name="weights")
y_model = model(X, w)
cost = tf.square(Y - y_model)
train_op = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

for i in range(100):
    for (x, y) in zip(trX, trY):
        sess.run(train_op, feed_dict={X: x, Y: y})
        print(sess.run(w))
```

A separate, Turing complete, virtual machine.



Computation Graph

DECLARATIVE TOOLKITS

- Declare a computation
- Placeholder variables
- Compile it
- Run it in a Session

ADVANTAGES

End-to-end program optimization
Easy to serialize for production deployment

DISADVANTAGES

Non-intuitive programming model
Difficult to design and maintain



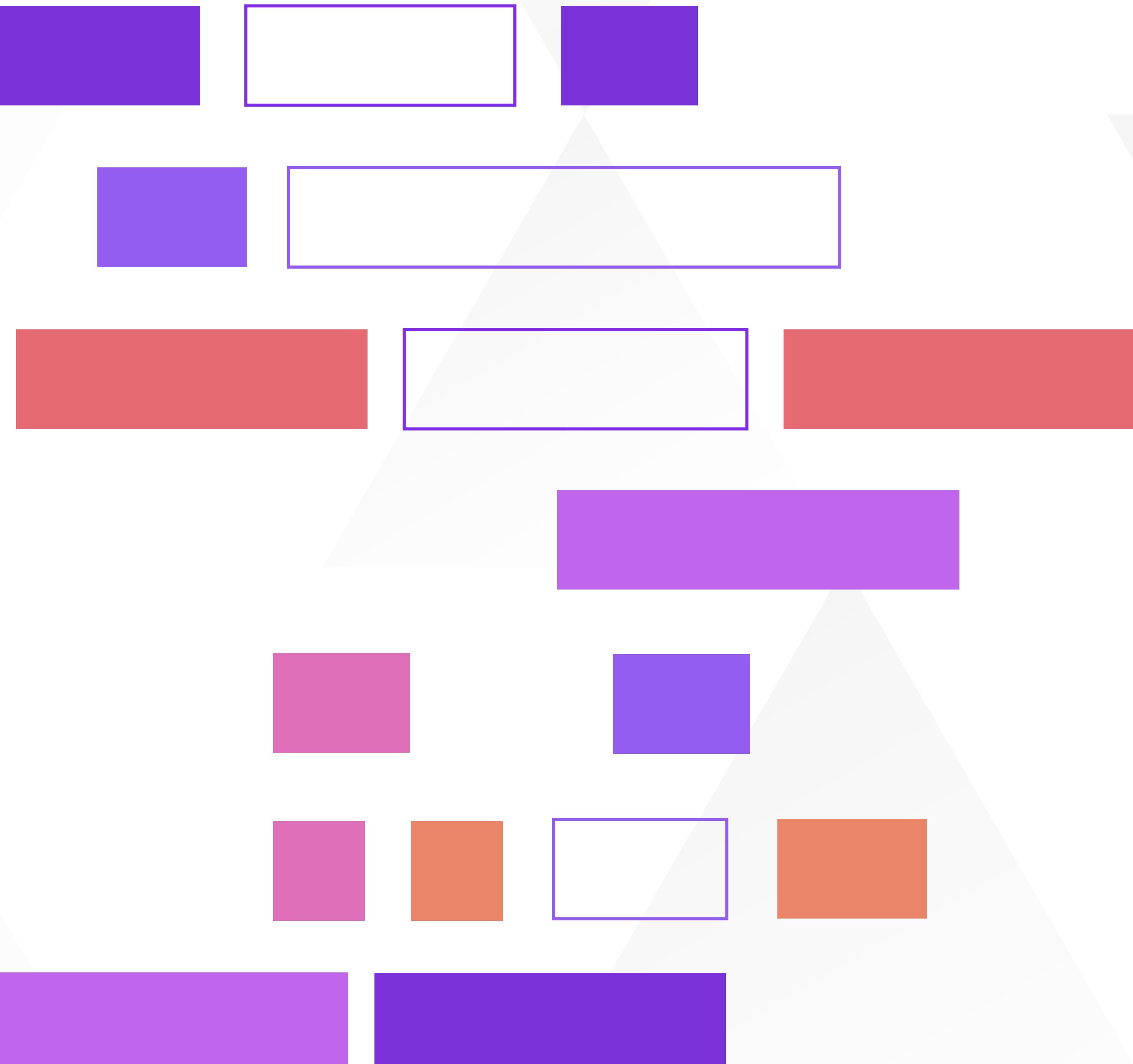
DEFINE-BY-RUN

Imperative Toolkits



P Y T  R C H

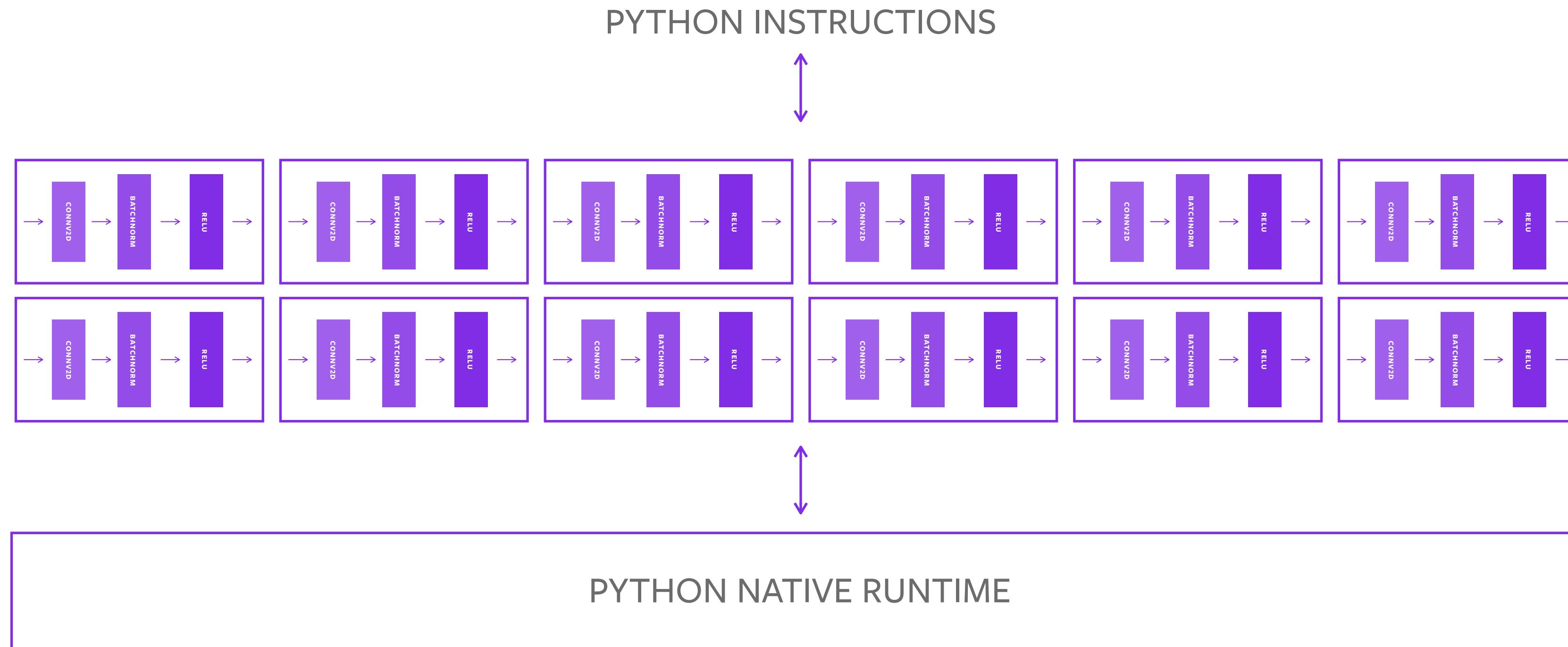
$\partial y / \text{net}$





IMPERATIVE TOOLKITS

Run a series of computation
Implicitly defining the model as execution goes





Imperative Toolkits

- Define a model by execution
- No separate compilation stage
- No separate execution engine

```
import torch
from torch.autograd import Variable

trX = torch.linspace(-1, 1, 101)
trY = 2 * trX + torch.random(*trX.size()) * 0.33

w = Variable(trX.new([0.0]), requires_grad=True)

for i in range(100):
    for (x, y) in zip(trX, trY):
        X = Variable(x)
        Y = Variable(y)
        print(X)
        print(Y)

        y_model = X * w.expand_as(X)
        cost = (Y - y_model) ** 2
        Cost.backward(torch.ones(*cost.size()))

        w.data = w.data + 0.01 * w.grad.data

        print(w)
```



Imperative Toolkits

- Define a model by execution
- No separate compilation stage
- No separate execution engine

Model constructed and values computed as we define it.

```
trX = torch.linspace(-1, 1, 101)
trY = 2 * trX + torch.random(*trX.size()) * 0.33

w = Variable(trX.new([0.0]), requires_grad=True)

for i in range(100):
    for (x, y) in zip(trX, trY):
        X = Variable(x)
        Y = Variable(y)
        print(X)
        print(Y)

        y_model = X * w.expand_as(X)
        cost = (Y - y_model) ** 2
        Cost.backward(torch.ones(*cost.size()))

        w.data = w.data + 0.01 * w.grad.data

        print(w)
```



Imperative Toolkits

- Define a model by execution
- No separate compilation stage
- No separate execution engine

ADVANTAGES

Natural, flexible program model
Ability to use Python ecosystem
Easy to debug

DISADVANTAGES

Difficult to optimize
No clear model serialization
Overhead can be significant



Bridging Imperative and Declarative



PyTorch *Eager Mode*

Models are Python program,
autograd for derivatives

- + Simple
- + Debuggable — `print` and `pdb`
- + Hackable — use any Python library
- Needs Python to run
- Difficult to optimize and parallelize

PyTorch *Script Mode*

Models are programs written in
an optimizable subset of Python

- + Production deployment
- + No Python dependency
- + Optimizable



P Y T O R C H J I T

Tools to transition eager code into script mode

EAGER
MODE

For prototyping, training,
and experiments

`@torch.jit.script`



`torch.jit.trace`

SCRIPT
MODE

For use at scale
in production

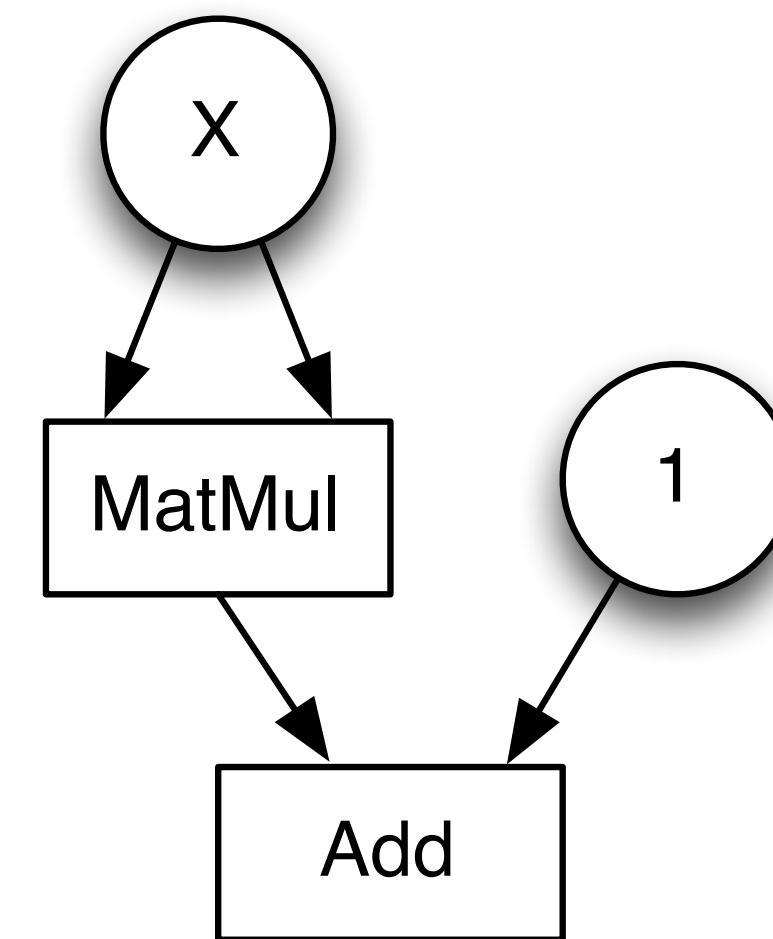


Tracing

```
def foo(x, t):  
    y = x.mm(x)  
    print(y) # still works!  
    return y + t
```

```
x = torch.Tensor([[1,2],[3,4]])  
foo(x, 1)
```

```
trace = torch.jit.trace(foo, (x, 1))  
trace.save("serialized.pt")
```



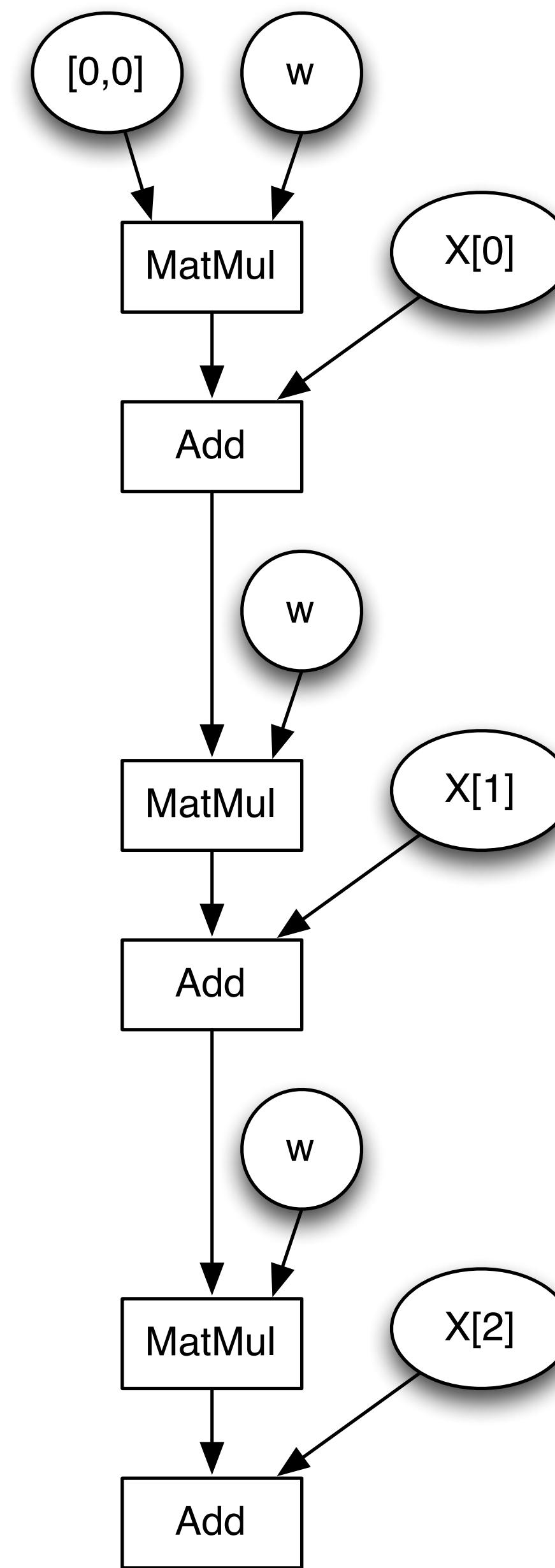


Tracing

```
def foo(x, t):
    y = x.mm(x)
    print(y) # still works!
    return y + t
```

```
def bar(x, w):
    y = torch.zeros(1, 2)
    for t in x:
        y = foo(y, w, t)
    return y
```

```
trace = torch.jit.trace(foo, (x, 1))
trace.save("serialized.pt")
```





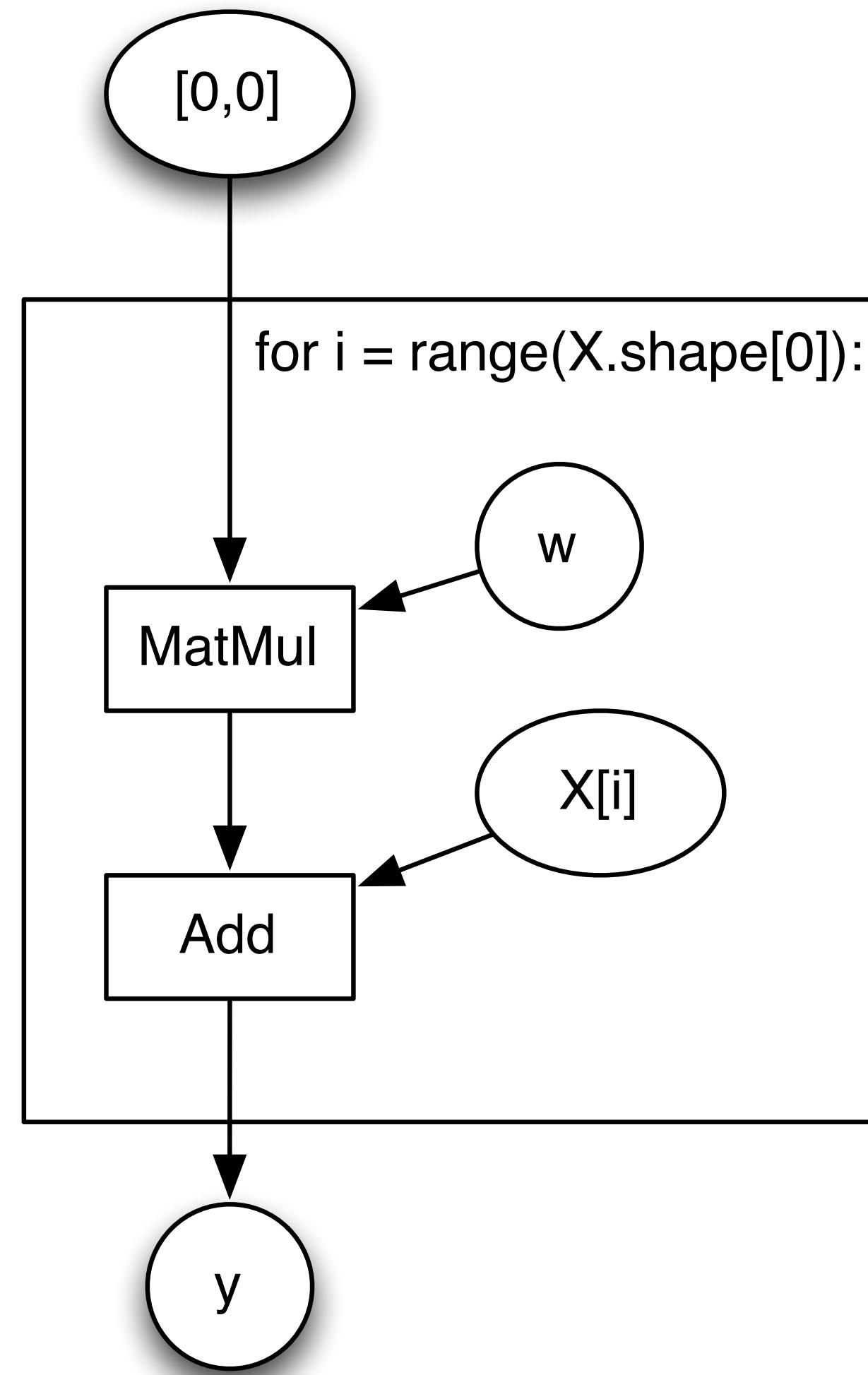
Script

```
def foo(x, t):
    y = x.mm(x)
    print(y) # still works!
    return y + t
```

@script

```
def bar(x, w):
    y = torch.zeros(1, 2)
    for t in x:
        y = foo(y, w, t)
    return y
```

```
trace = torch.jit.trace(foo, (x, 1))
trace.save("serialized.pt")
```





Transitioning a model with `@torch.jit.script`

Write model directly in a subset of Python,
annotated with `@torch.jit.script` or
`@torch.jit.scri`

- Control-flow is preserved
- `print` statements for debugging
- Remove the annotations with standard Python tools.

```
class RNN(torch.jit.ScriptModule):
    def __init__(self, W_h, U_h, W_y, b_h, b_y):
        super().__init__()
        self.W_h = W_h
        self.U_h = U_h
        self.W_y = W_y
        self.b_h = b_h
        self.b_y = b_y

    @torch.jit.script
    def forward(self, x, h):
        y = []
        for t in range(x.size(0)):
            h = torch.tanh(x[t] @ self.W_h + h @ self.U_h + self.b_h)
            y += [torch.tanh(h @ self.W_y + self.b_y)]
            if t % 10 == 0:
                print("stats: ", h.mean(), h.var())
        return torch.stack(y), h
```

You can mix both trace and script
in a single model.



Loading a model without Python

Torch Script models can be saved to a model archive, and loaded in a python-free executable using a C++ API.

Our C++ Tensor API is the same as our Python API, so you can do preprocessing and post processing before calling the model.

```
# Python: save model
traced_resnet = torch.jit.trace(torchvision.models.resnet18(),
                                torch.rand(1, 3, 224, 224))
traced_resnet.save("serialized_resnet.pt")
```

```
// C++: load and run model
auto module = torch::jit::load("serialized_resnet.pt");
auto example = torch::rand({1, 3, 224, 224});
auto output = module->forward({example}).toTensor();
std::cout << output.slice(1, 0, 5) << '\n';
```



P R E V I E W R E L E A S E

Try the PyTorch JIT today

I N S T A L L

Using the nightly preview builds of PyTorch.

START LOCALLY

Select your preferences and run the install command. Please ensure that you are on the latest pip and numpy packages. Anaconda is our recommended package manager. You can also [Install previous versions of PyTorch](#).

PyTorch Build: Stable | Preview

Your OS: Linux | Mac | Windows

Package: Conda | Pip | Source

Python: 2.7 | 3.5 | 3.6 | 3.7

CUDA: 8.0 | 9.0 | 9.2 | None

Run this Command: `conda install pytorch-nightly -c pytorch`

T U T O R I A L S

How to use tracing and script to export a sequence to sequence model.

PyTorch

Get Started Features Ecosystem Blog Tutorials Docs Resources GitHub

1.0.0.dev20180918 Tutorials Search Tutorials

WELCOME TO PYTORCH TUTORIALS

To learn how to use PyTorch, start with our Getting Started tutorial. The [60-minute blitz](#) is the most common starting point, and provides a broadview into how to use PyTorch from basics all the way to constructing deep neural networks.

Some considerations:

- If you would like to do the tutorials interactively via Jupyter / Juptyer
- Each tutorial has a download link for a Jupyter notebook and Python source code.
- Additional high-quality examples are available including image classification, unsupervised learning, reinforcement learning, machine translation and many other applications at [https://github.com/pytorch/examples/](https://github.com/pytorch/examples)
- You can find reference documentation for PyTorch's API and layers at <http://docs.pytorch.org> or via inline help.
- If you would like the tutorials section improved, please open a GitHub issue.

Here with your feedback: <https://github.com/pytorch/tutorials>

Only some of the tutorials are marked as requiring the Preview release. These are tutorials that use the new

D O C S

API and language reference for Torch Script.

PyTorch

Get Started Features Ecosystem Blog Tutorials Docs Resources GitHub

master (1.0.0dev20180918) Search Docs

You are viewing unstable developer preview docs. Click here to view docs for stable stable release.

Notes

- Autograd mechanics
- Broadcasting semantics
- CUDA semantics
- Extending PyTorch
- Frequently Asked Questions
- Multiprocessing best practices
- Reproducibility
- Semilocation semantics
- Windows FAQ

Package Reference

- torch
- torchTensor
- Tensor Attributes
- torchSparse
- torchCuda
- torchStorage

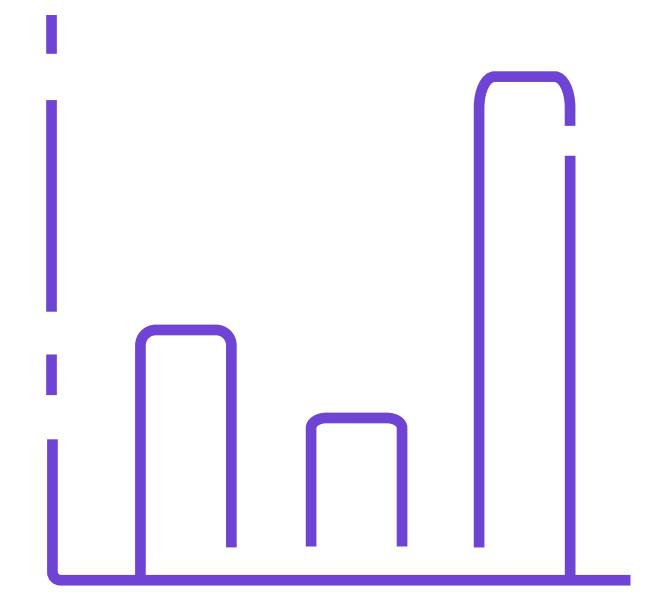
PyTorch Documentation

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

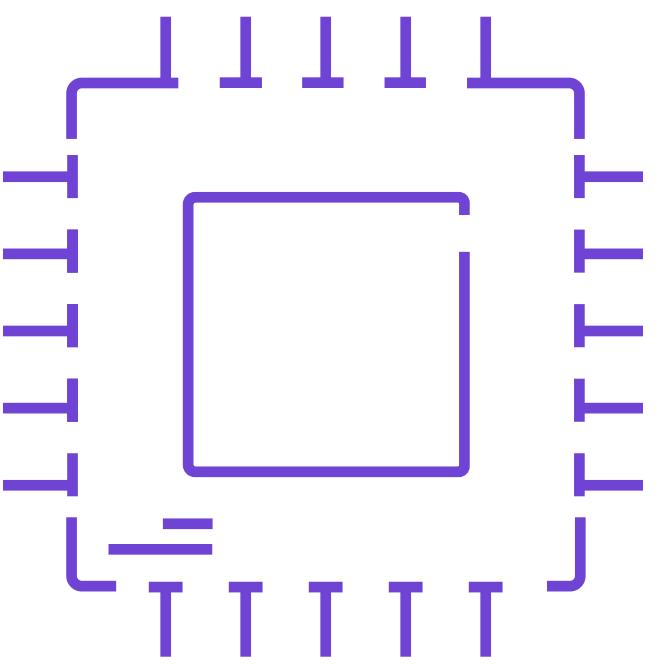


Scale

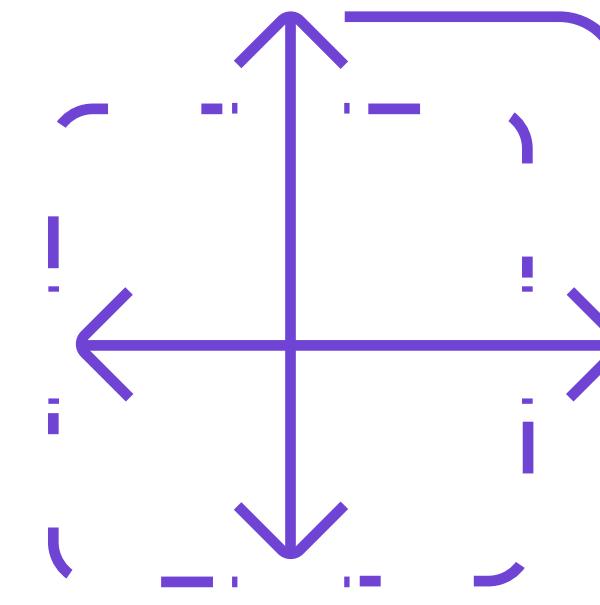




MORE DATA



FASTER COMPUTE



MODEL EXPLORATION





JUN 8, 2017 - Facebook

Accelerating machine learning for computer vision

<https://code.fb.com/core-data/accelerating-machine-learning-for-computer-vision/>

MAY 2, 2018 - Facebook

Advancing state-of-the-art image recognition with deep learning on hashtags

<https://code.fb.com/ml-applications/advancing-state-of-the-art-image-recognition-with-deep-learning-on-hashtags/>

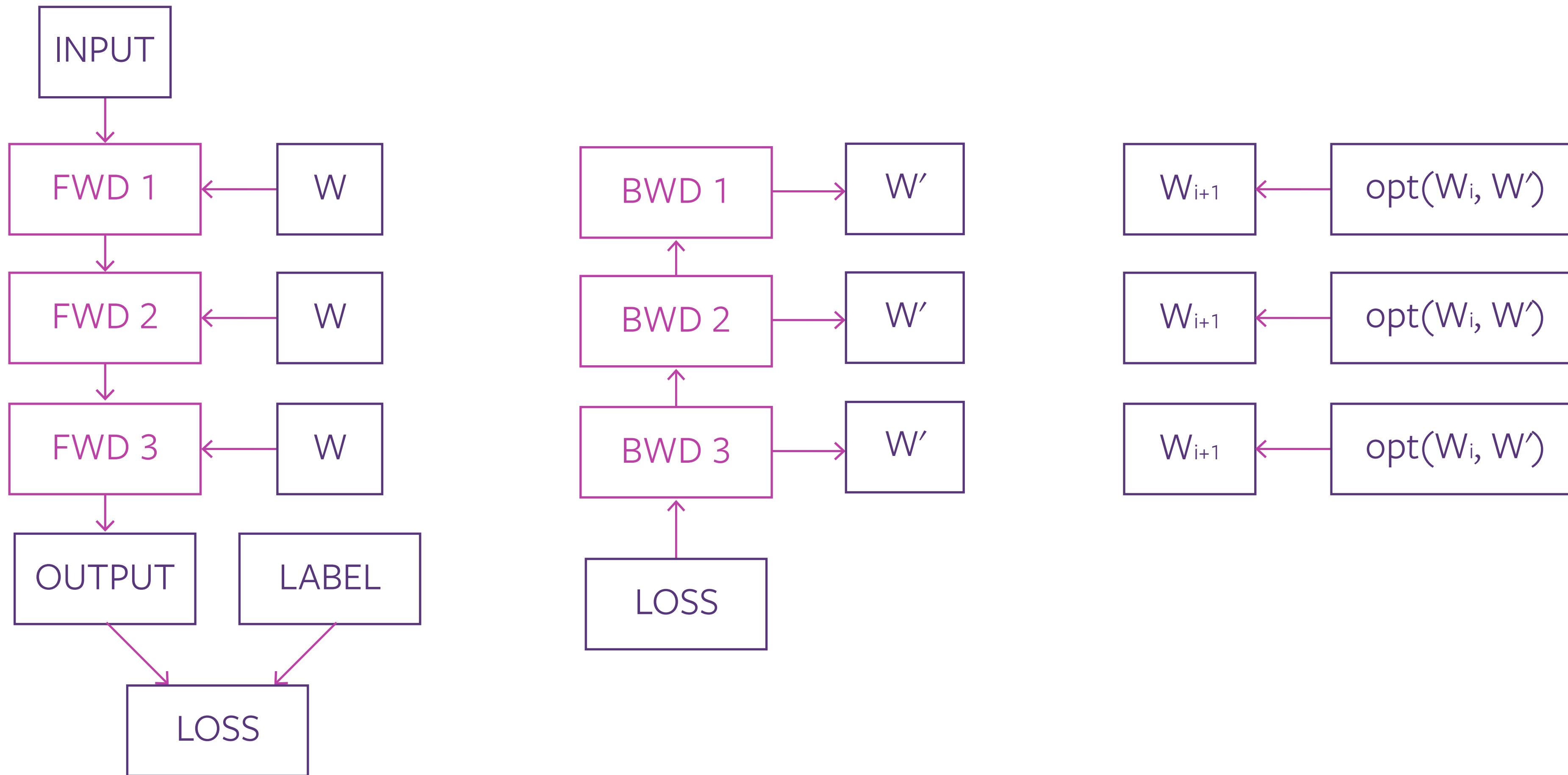
AUG 10, 2018 - fast.ai

Now anyone can train ImageNet in 18 minutes

<http://www.fast.ai/2018/08/10/fastai-diu-imagenet/>



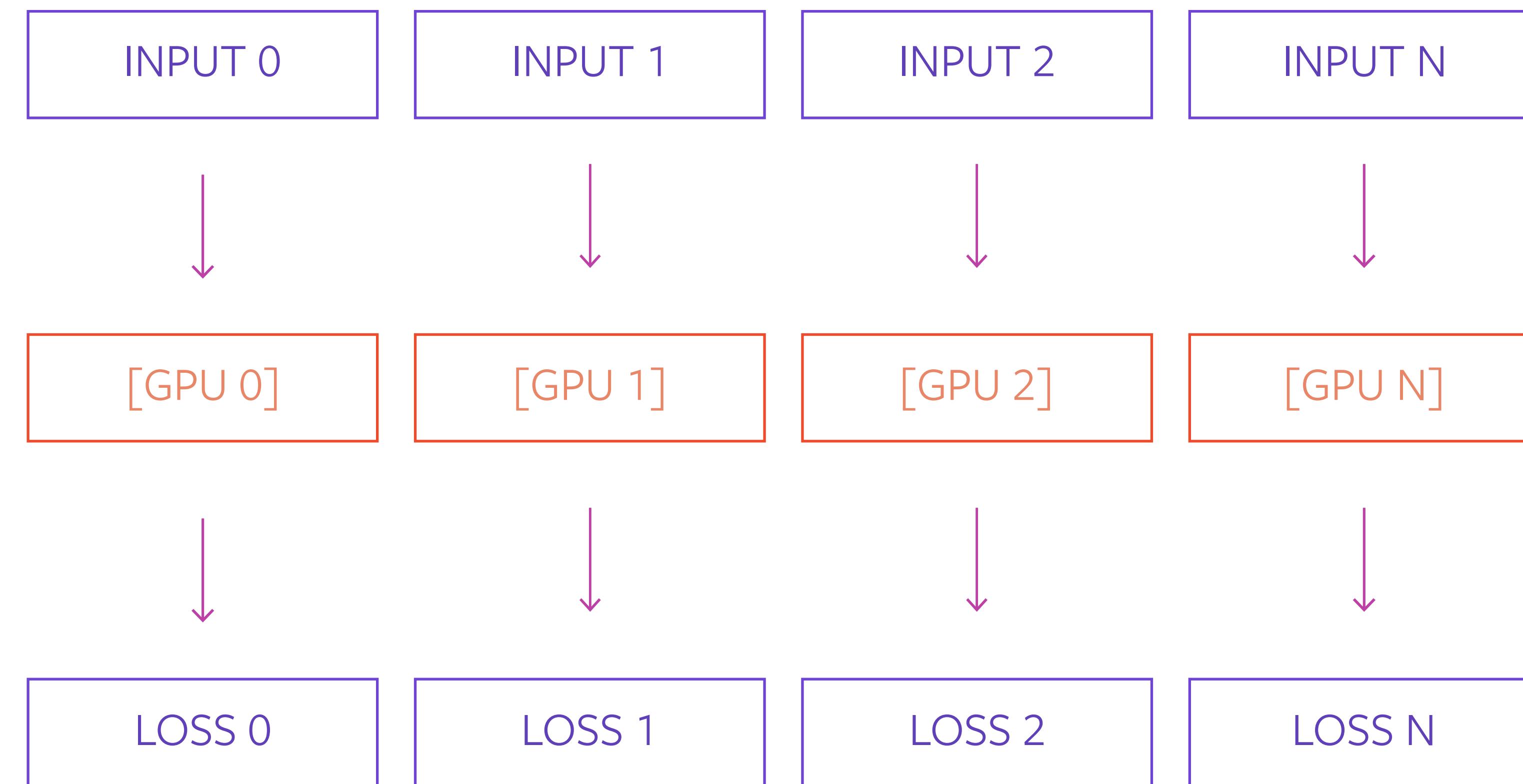
Anatomy of a single iteration





Data parallelism

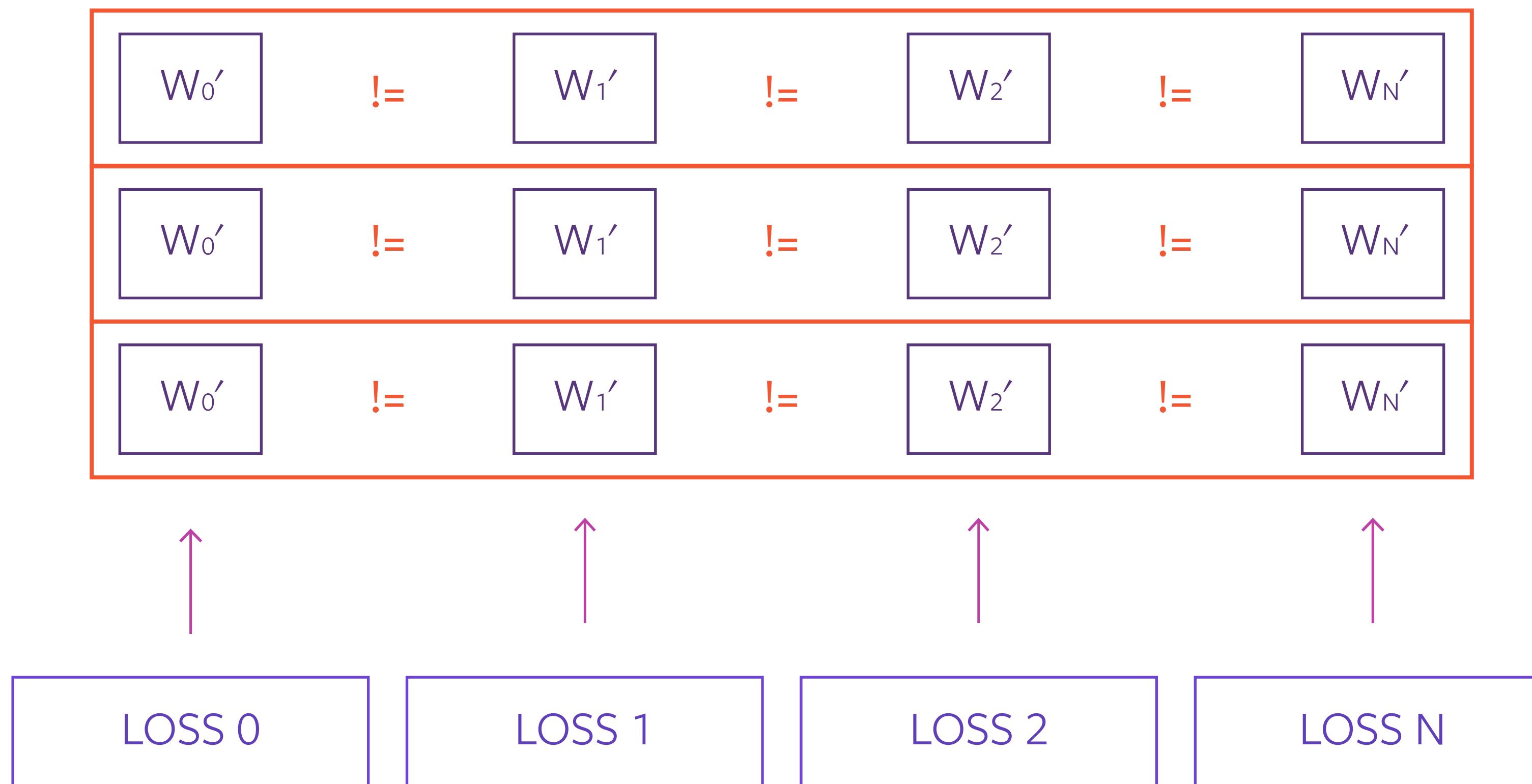
FORWARD PASS





Data parallelism

BACKWARD PASS





Data parallelism

AVERAGE GRADIENTS

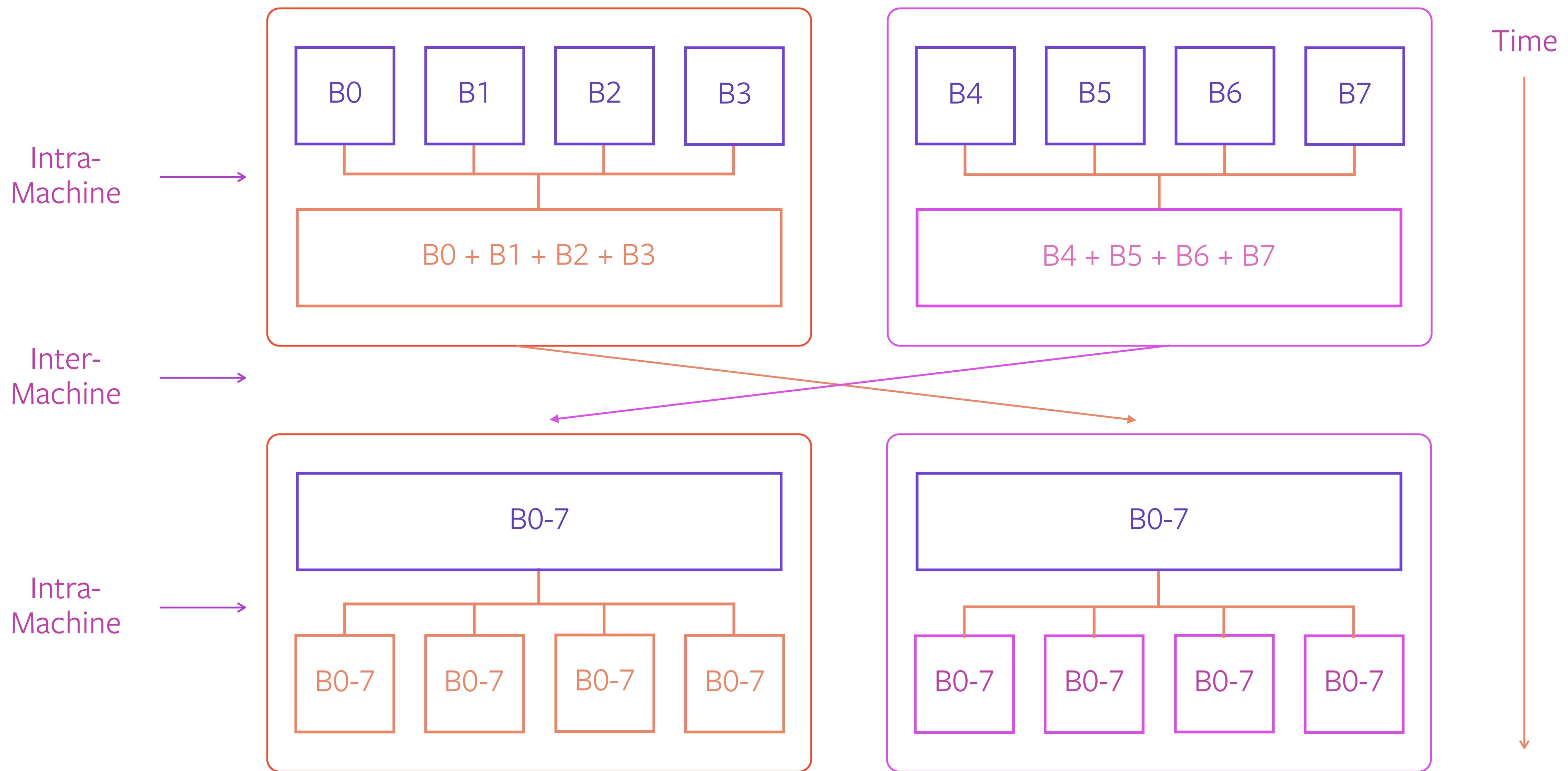
$$W' = (W_0' + W_1' + W_2' + \dots + W_N') / N$$

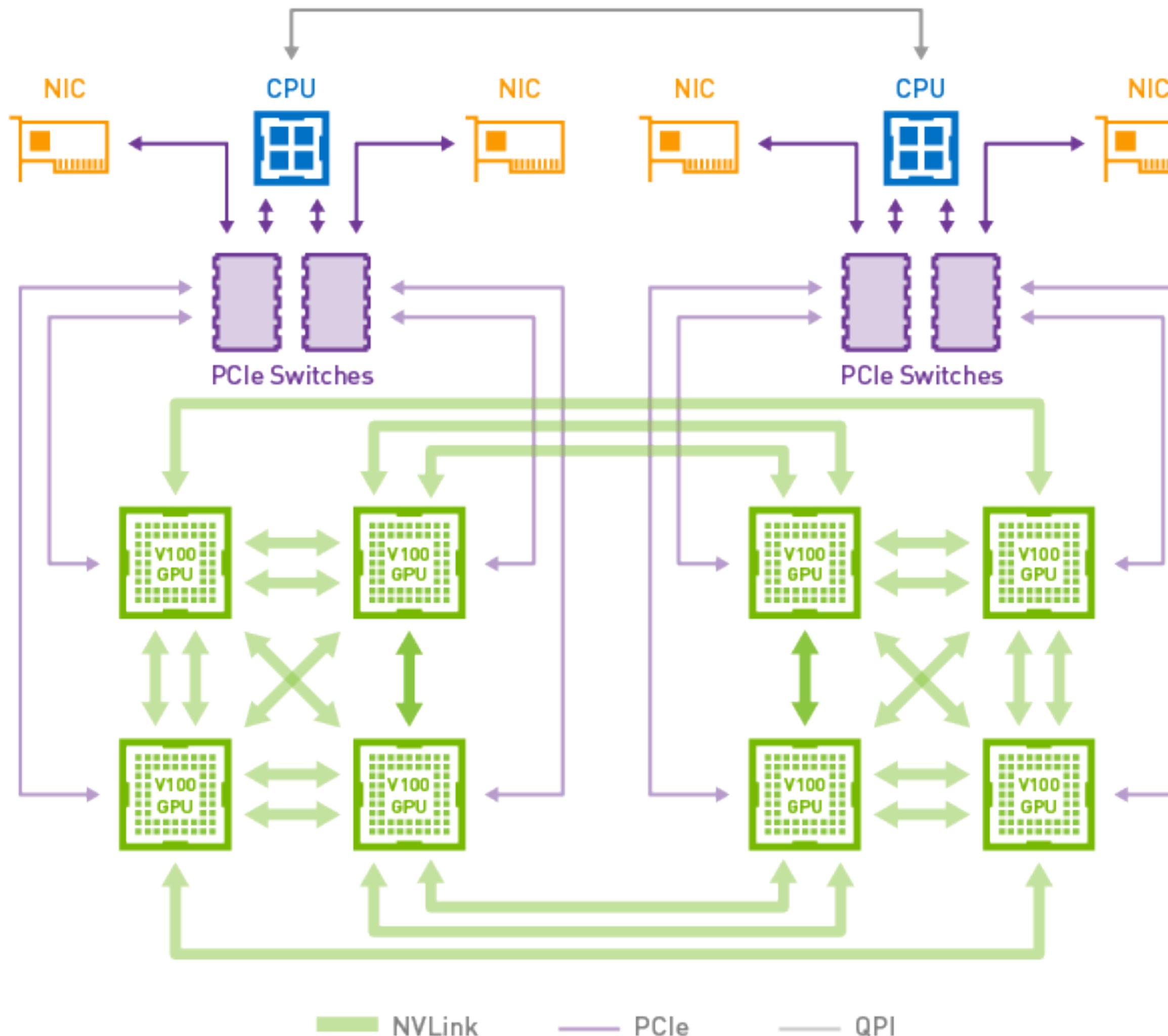
$$W' = (W_0' + W_1' + W_2' + \dots + W_N') / N$$

$$W' = (W_0' + W_1' + W_2' + \dots + W_N') / N$$



The maximum achievable speedup
is determined by the overhead
of the allreduce operation



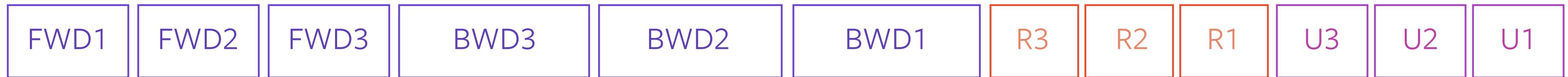
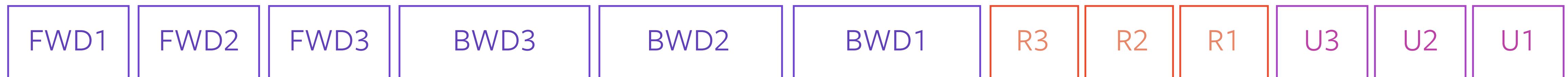


NVLink connecting eight Tesla V100 accelerators in a hybrid cube mesh topology as used in the DGX-1V server
<https://www.nvidia.com/en-us/data-center/nvlink/>



Data parallelism

OVERLAPPING BACKWARD PASS WITH ALLREDUCE





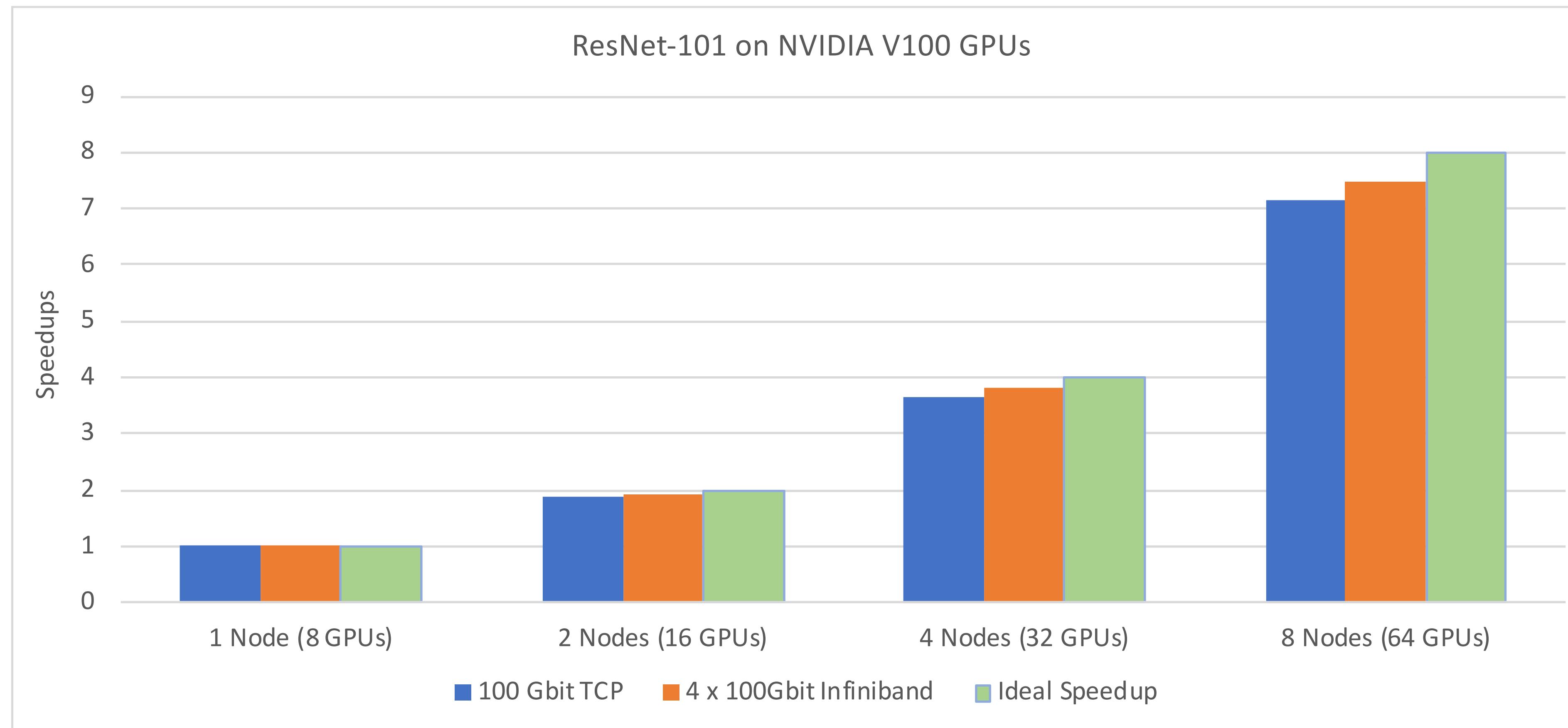
Data parallelism

OVERLAPPING BACKWARD PASS WITH ALLREDUCE





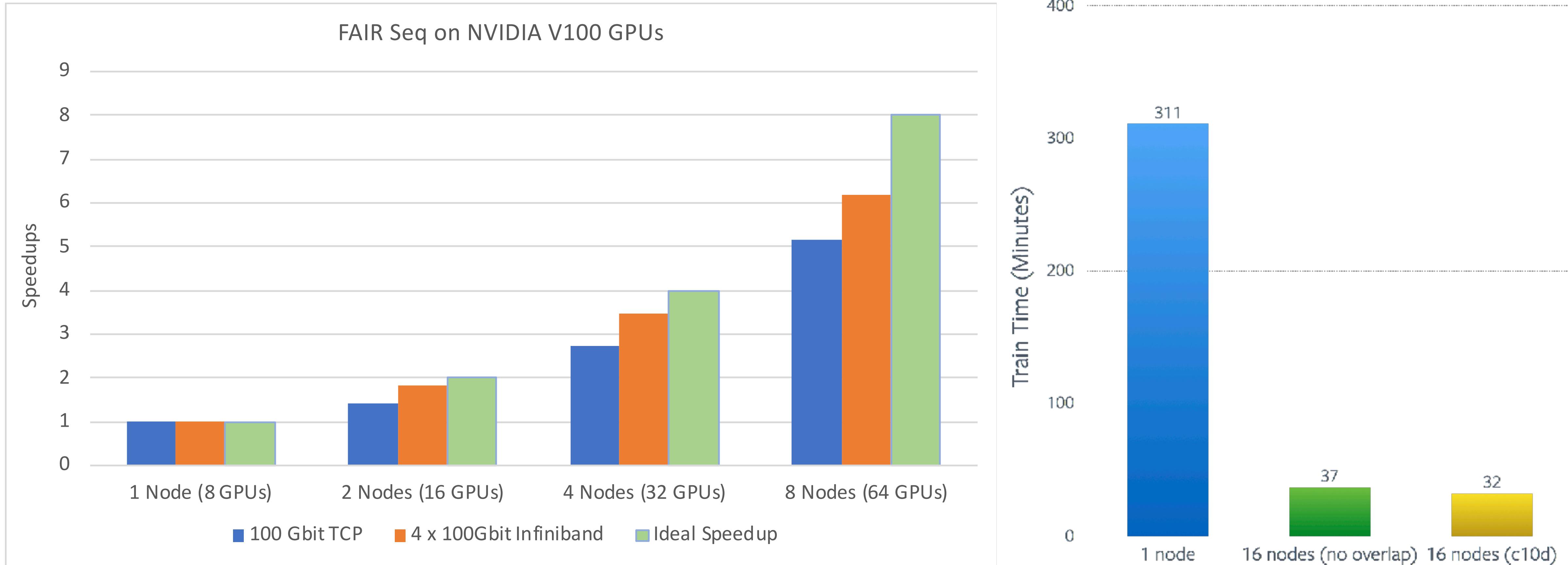
Distributed Training Performance – ResNet101





Distributed Training Performance – FAIR Seq

Bonjour à tous ! → Hello everybody!



- 311 minutes – 32 minutes, by going from 1 to 16 NVIDIA DGX-1 nodes (8 to 128 NVIDIA V100 GPUs)
- 19% performance gain (1.53M – 1.82M Words Per Second on 16 nodes), thanks to c10d DDP overlapping



Data Parallelism API

```
for epoch in range(max_epochs):
    for data, target in enumerate(training_data):
        output = model(data)
        loss = F.nll_loss(output, target)
        loss.backward()
        optimizer.step()
```



Data Parallelism API

```
torch.distributed.init_process_group(world_size=4, init_method='...')  
model = torch.nn.DistributedDataParallel(model)  
  
for epoch in range(max_epochs):  
    for data, target in enumerate(training_data):  
        output = model(data)  
        loss = F.nll_loss(output, target)  
        loss.backward()  
        optimizer.step()
```



PyTorch Ecosystem

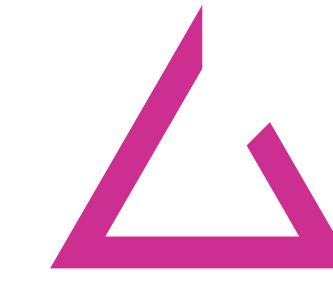


PYTORCH ECOSYSTEM

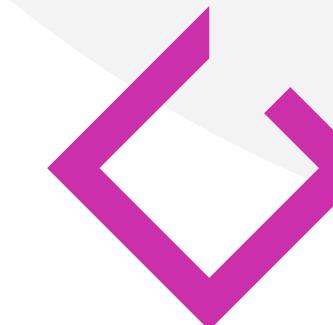
Tools, Libraries and Datasets
to enable cutting-edge AI
development



PyTorch **Vision**



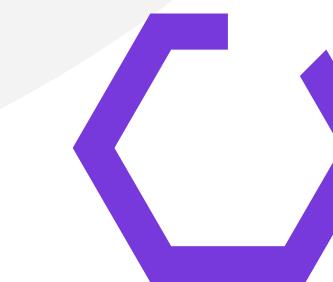
PyTorch **Reasoning**



PyTorch **Language**



PyTorch **Speech**



PyTorch **Tools**

Ecosystem

- Use the entire Python ecosystem at your will



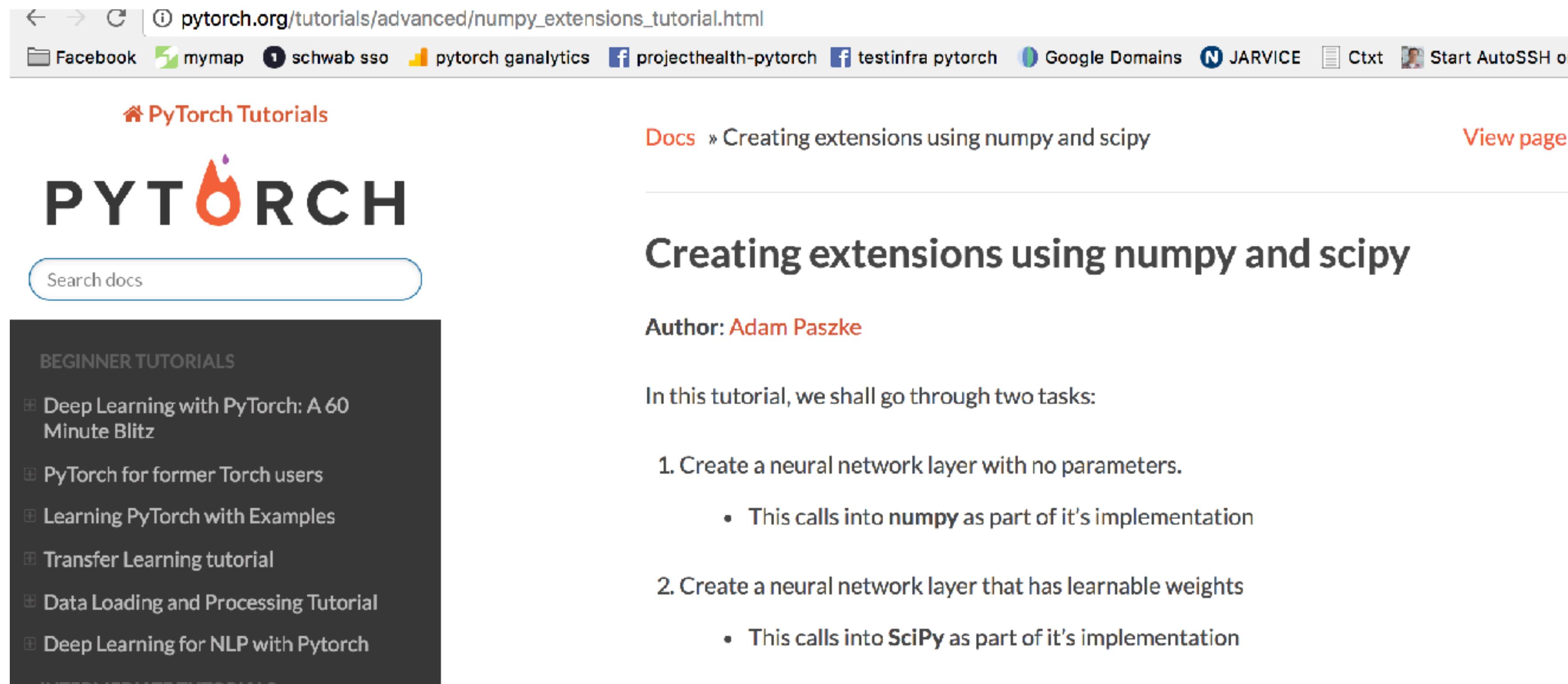
Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



Ecosystem

- Use the entire Python ecosystem at your will
- Including SciPy, Scikit-Learn, etc.



The screenshot shows a browser window displaying the PyTorch documentation. The URL in the address bar is `pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html`. The page title is "Creating extensions using numpy and scipy". The author is listed as Adam Paszke. The content describes two tasks: creating a neural network layer with no parameters (calling into NumPy) and creating one with learnable weights (calling into SciPy). On the left sidebar, there's a "BEGINNER TUTORIALS" section with links to "Deep Learning with PyTorch: A 60 Minute Blitz", "PyTorch for former Torch users", "Learning PyTorch with Examples", "Transfer Learning tutorial", "Data Loading and Processing Tutorial", and "Deep Learning for NLP with Pytorch".

← → C | i pytorch.org/tutorials/advanced/numpy_extensions_tutorial.html

Facebook mymap schwab sso pytorch ganalytics projecthealth-pytorch testinfra pytorch Google Domains JARVICE Ctxt Start AutoSSH or

PyTorch Tutorials

Docs » Creating extensions using numpy and scipy

View page

PYTORCH

Search docs

BEGINNER TUTORIALS

- Deep Learning with PyTorch: A 60 Minute Blitz
- PyTorch for former Torch users
- Learning PyTorch with Examples
- Transfer Learning tutorial
- Data Loading and Processing Tutorial
- Deep Learning for NLP with Pytorch

INTERMEDIATE TUTORIALS

Creating extensions using numpy and scipy

Author: Adam Paszke

In this tutorial, we shall go through two tasks:

1. Create a neural network layer with no parameters.
 - This calls into `numpy` as part of its implementation
2. Create a neural network layer that has learnable weights
 - This calls into `SciPy` as part of its implementation



Ecosystem

- A shared model-zoo:

We provide pre-trained models for the ResNet variants and AlexNet, using the PyTorch `torch.utils.model_zoo`. These can be constructed by passing `pretrained=True`:

```
import torchvision.models as models  
resnet18 = models.resnet18(pretrained=True)  
alexnet = models.alexnet(pretrained=True)
```



Ecosystem

- A shared

GitHub, Inc. [US] | https://github.com/aaron-xichen/pytorch-playground

mymap schwab sso pytorch ganalytics projecthealth-pytorch testinfra pytorch Google Domains JARVICE Ctxt Start Auto

We evaluate the performance of popular dataset and models with linear quantized method. The bit-width of running mean and running variance in BN are 10 bits for all results. (except for 32-float)

| Model | 32-float | 12-bit | 10-bit | 8-bit | 6-bit |
|--------------|-------------|-------------|-------------|-------------|-------------|
| MNIST | 98.42 | 98.43 | 98.44 | 98.44 | 98.32 |
| SVHN | 96.03 | 96.03 | 96.04 | 96.02 | 95.46 |
| CIFAR10 | 93.78 | 93.79 | 93.80 | 93.58 | 90.86 |
| CIFAR100 | 74.27 | 74.21 | 74.19 | 73.70 | 66.32 |
| STL10 | 77.59 | 77.65 | 77.70 | 77.59 | 73.40 |
| AlexNet | 55.70/78.42 | 55.66/78.41 | 55.54/78.39 | 54.17/77.29 | 18.19/36.25 |
| VGG16 | 70.44/89.43 | 70.45/89.43 | 70.44/89.33 | 69.99/89.17 | 53.33/76.32 |
| VGG19 | 71.36/89.94 | 71.35/89.93 | 71.34/89.88 | 70.88/89.62 | 56.00/78.62 |
| ResNet18 | 68.63/88.31 | 68.62/88.33 | 68.49/88.25 | 66.80/87.20 | 19.14/36.49 |
| ResNet34 | 72.50/90.86 | 72.46/90.82 | 72.45/90.85 | 71.47/90.00 | 32.25/55.71 |
| ResNet50 | 74.98/92.17 | 74.94/92.12 | 74.91/92.09 | 72.54/90.44 | 2.43/5.36 |
| ResNet101 | 76.69/93.30 | 76.66/93.25 | 76.22/92.90 | 65.69/79.54 | 1.41/1.18 |
| ResNet152 | 77.55/93.59 | 77.51/93.62 | 77.40/93.54 | 74.95/92.46 | 9.29/16.75 |
| SqueezeNetV0 | 56.73/79.39 | 56.75/79.40 | 56.70/79.27 | 53.93/77.04 | 14.21/29.74 |
| SqueezeNetV1 | 56.52/79.13 | 56.52/79.15 | 56.24/79.03 | 54.56/77.33 | 17.10/32.46 |
| InceptionV3 | 76.41/92.78 | 76.43/92.71 | 76.44/92.73 | 73.67/91.34 | 1.50/4.82 |



fast.ai 1.0

High-level library on PyTorch: <http://docs.fast.ai>

- state-of-the-art models in few lines
- fine-tune on your own data

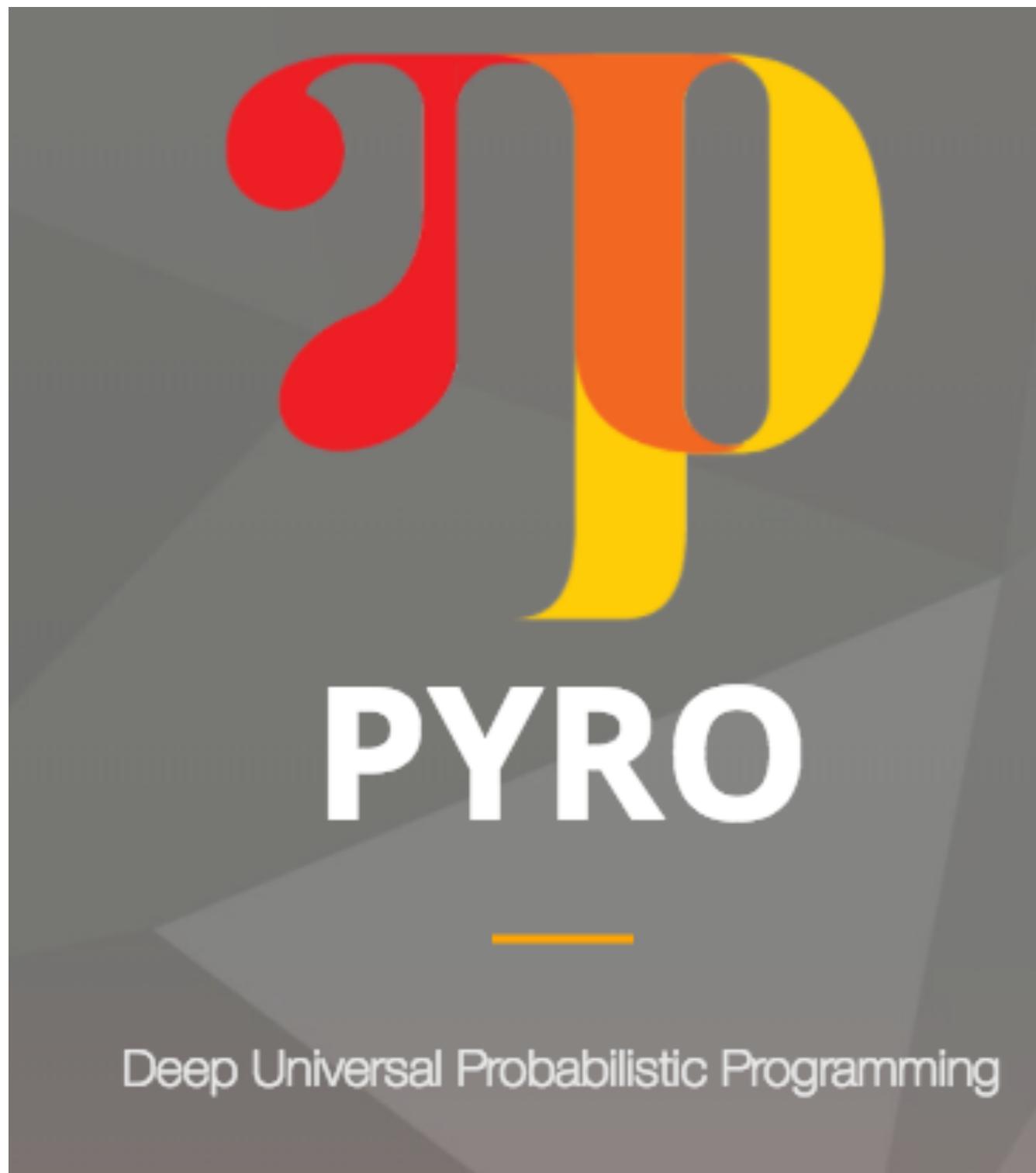
Built by Jeremy Howard, Rachel Thomas and community
+ online course in addition to the library

Read more at <http://www.fast.ai/2018/10/02/fastai-ai/>



Ecosystem

- Probabilistic Programming



github.com/probtorch/probtorch

<http://pyro.ai/>



Ecosystem

• Gaussian Processes

GPyTorch (Alpha Release)

[build](#) [passing](#)

GPyTorch is a Gaussian Process library, implemented using PyTorch. It is designed for creating flexible and modular Gaussian Process models with ease, so that you don't have to be an expert to use GPs.

This package is currently under development, and is likely to change. Some things you can do right now:

- Simple GP regression ([example here](#))
- Simple GP classification ([example here](#))
- Multitask GP regression ([example here](#))
- Scalable GP regression using kernel interpolation ([example here](#))
- Scalable GP classification using kernel interpolation ([example here](#))
- Deep kernel learning ([example here](#))
- And ([more!](#))

<https://github.com/cornellius-gp/gpytorch>



Ecosystem

• Machine Translation

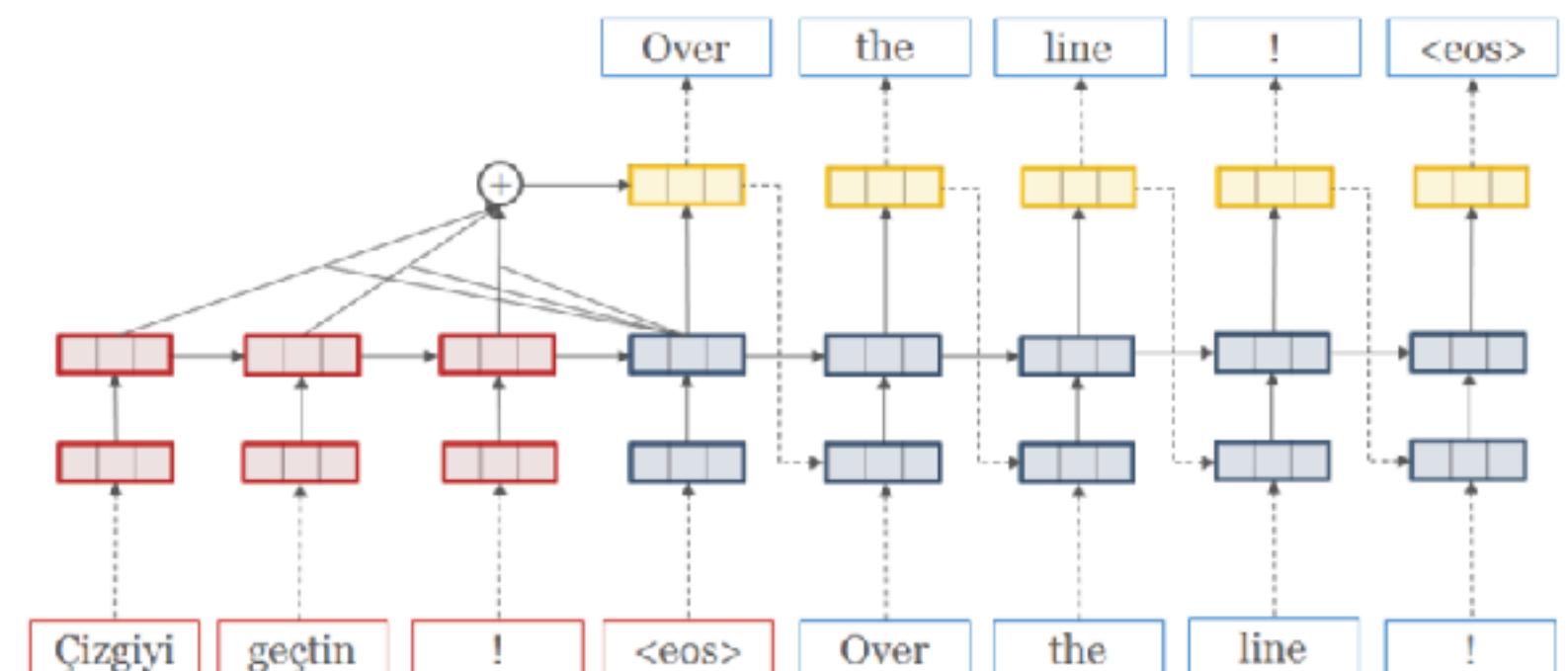
OpenNMT-py: Open-Source Neural Machine Translation

build passing

This is a Pytorch port of [OpenNMT](#), an open-source (MIT) neural machine translation system. It is designed to be research friendly to try out new ideas in translation, summary, image-to-text, morphology, and many other domains.

Codebase is relatively stable, but PyTorch is still evolving. We currently recommend forking if you need to have stable code.

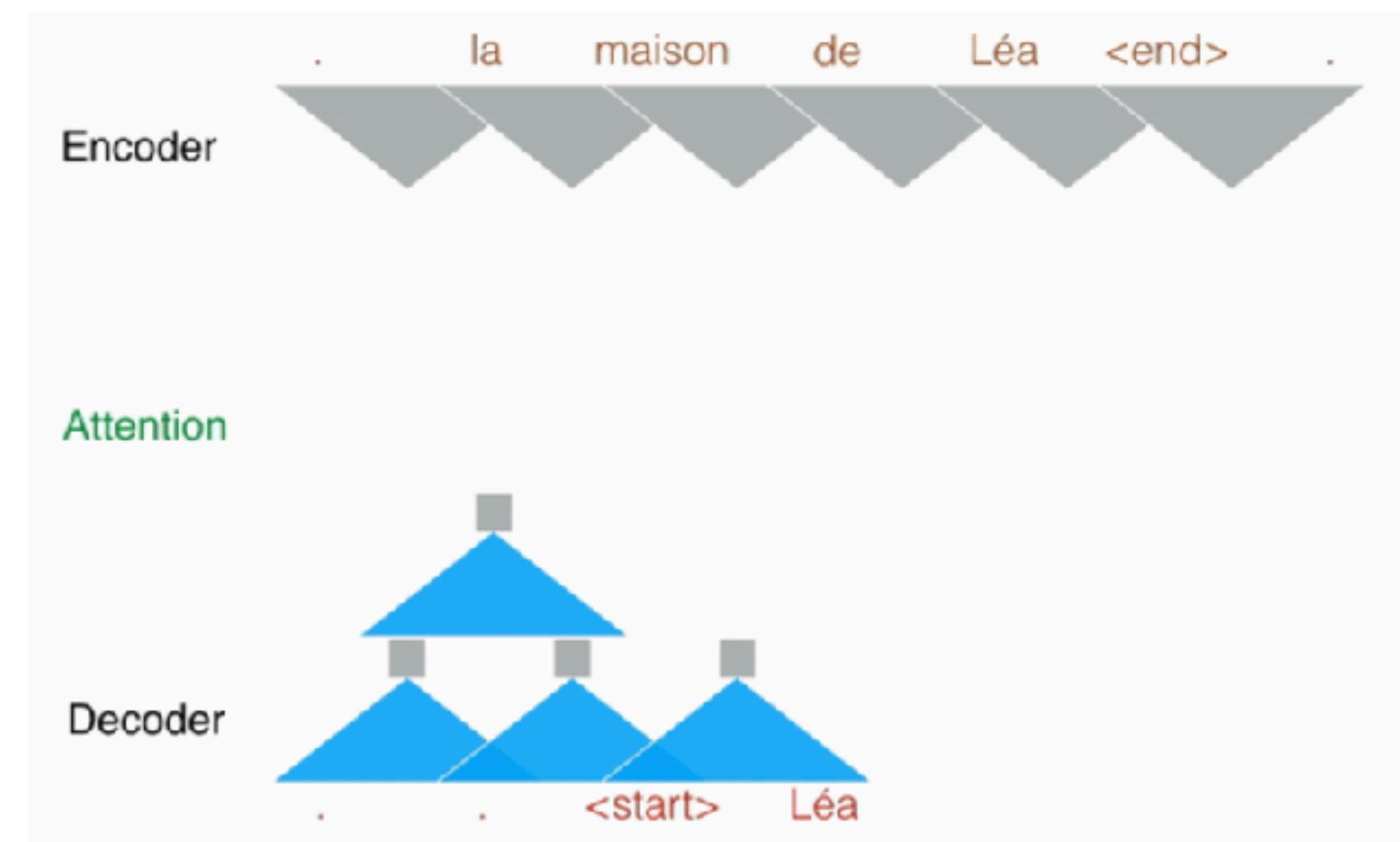
OpenNMT-py is run as a collaborative open-source project. It is maintained by [Sasha Rush](#) (Cambridge, MA), [Ben Peters](#) (Saarbrücken), and [Jianyu Zhan](#) (Shenzhen). The original code was written by [Adam Lerer](#) (NYC). We love contributions. Please consult the Issues page for any [Contributions Welcome](#) tagged post.



<https://github.com/OpenNMT/OpenNMT-py>

FAIR Sequence-to-Sequence Toolkit (PyTorch)

This is a PyTorch version of [fairseq](#), a sequence-to-sequence learning toolkit from Facebook AI Research. The original authors of this reimplementation are (in no particular order) Sergey Edunov, Myle Ott, and Sam Gross. The toolkit implements the fully convolutional model described in [Convolutional Sequence to Sequence Learning](#) and features multi-GPU training on a single machine as well as fast beam search generation on both CPU and GPU. We provide pre-trained models for English to French and English to German translation.



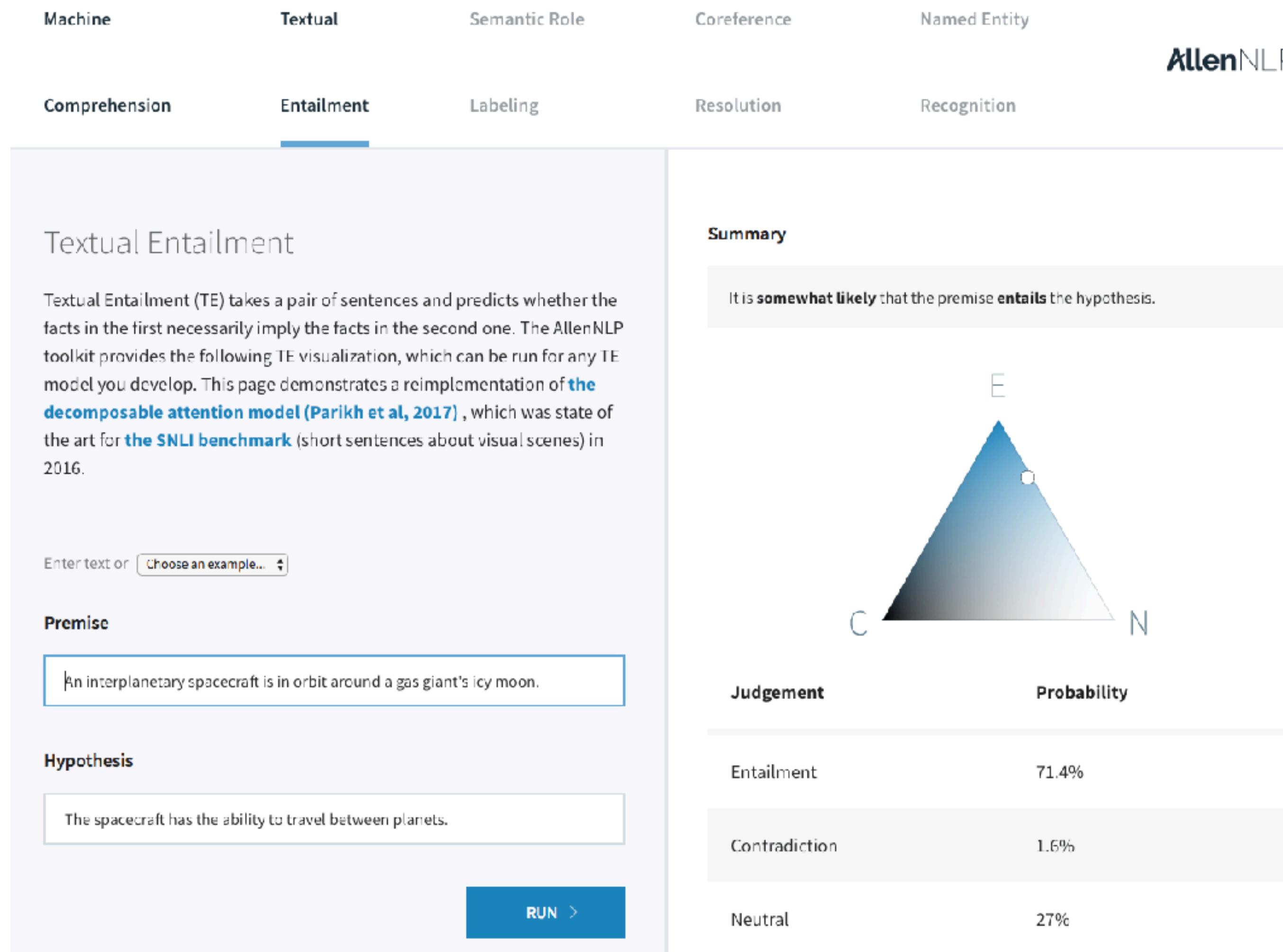
<https://github.com/facebookresearch/fairseq-py>



Ecosystem

- AllenNLP

<http://allennlp.org/>

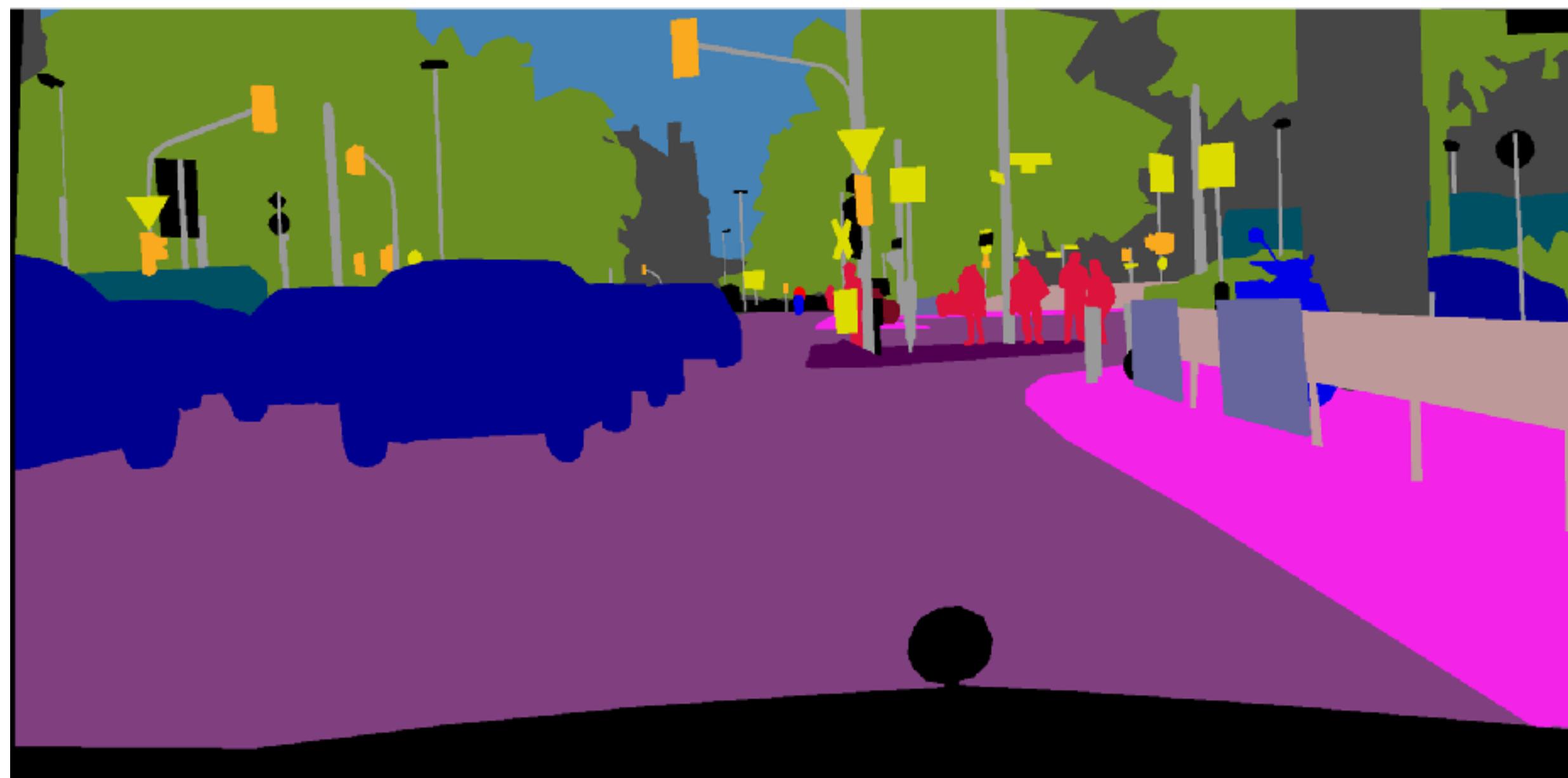


Ecosystem

- Pix2PixHD

<https://github.com/NVIDIA/pix2pixHD>

Input labels



Synthesized image



Ecosystem

- CycleGAN, pix2pix

<https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>





| pytorch.org



PyTorch 1.0

PRODUCTION BATTERIES INCLUDED



PyTorch 1.0

P R E V I E W



1 . 0 P R E V I E W

Available on pytorch.org

| | | | | | |
|-------------------|---|------------|------------|------------|-----|
| PyTorch Build | Stable | Preview | | | |
| Your OS | Linux | Mac | Windows | | |
| Package | Conda | Pip | LibTorch | Source | |
| Language | Python 2.7 | Python 3.5 | Python 3.6 | Python 3.7 | C++ |
| CUDA | 8.0 | 9.0 | 9.2 | None | |
| Run this Command: | <code>conda install pytorch-nightly -c pytorch</code> | | | | |



| pytorch.org