Bài 6:
Phần cứng và phần mềm cho học sâu

Nội dung

1. Phần cứng cho học sâu
2. Các nền tảng lập trình cho học sâu
3. Công cụ tăng tốc và nén mạng

Phần cứng cho học sâu

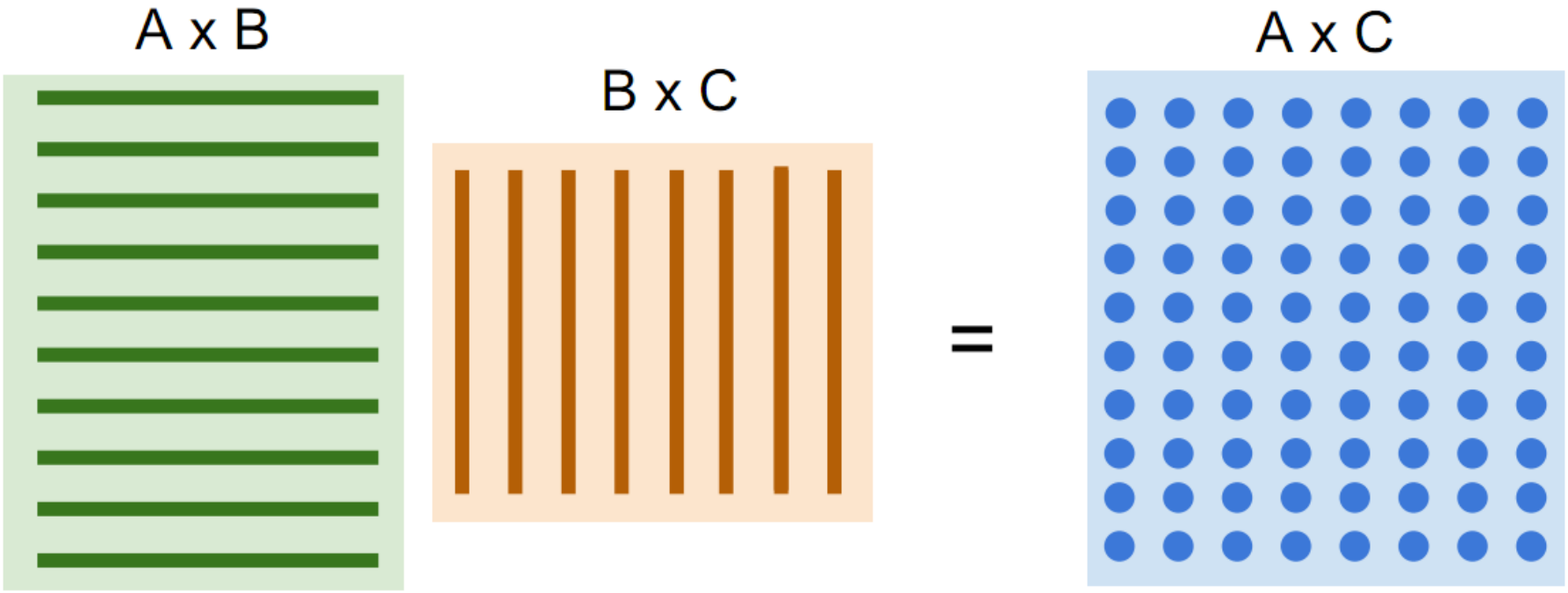
Một máy tính cho học sâu

CPU vs GPU

	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	$1199	~13.4 TFLOPs FP32

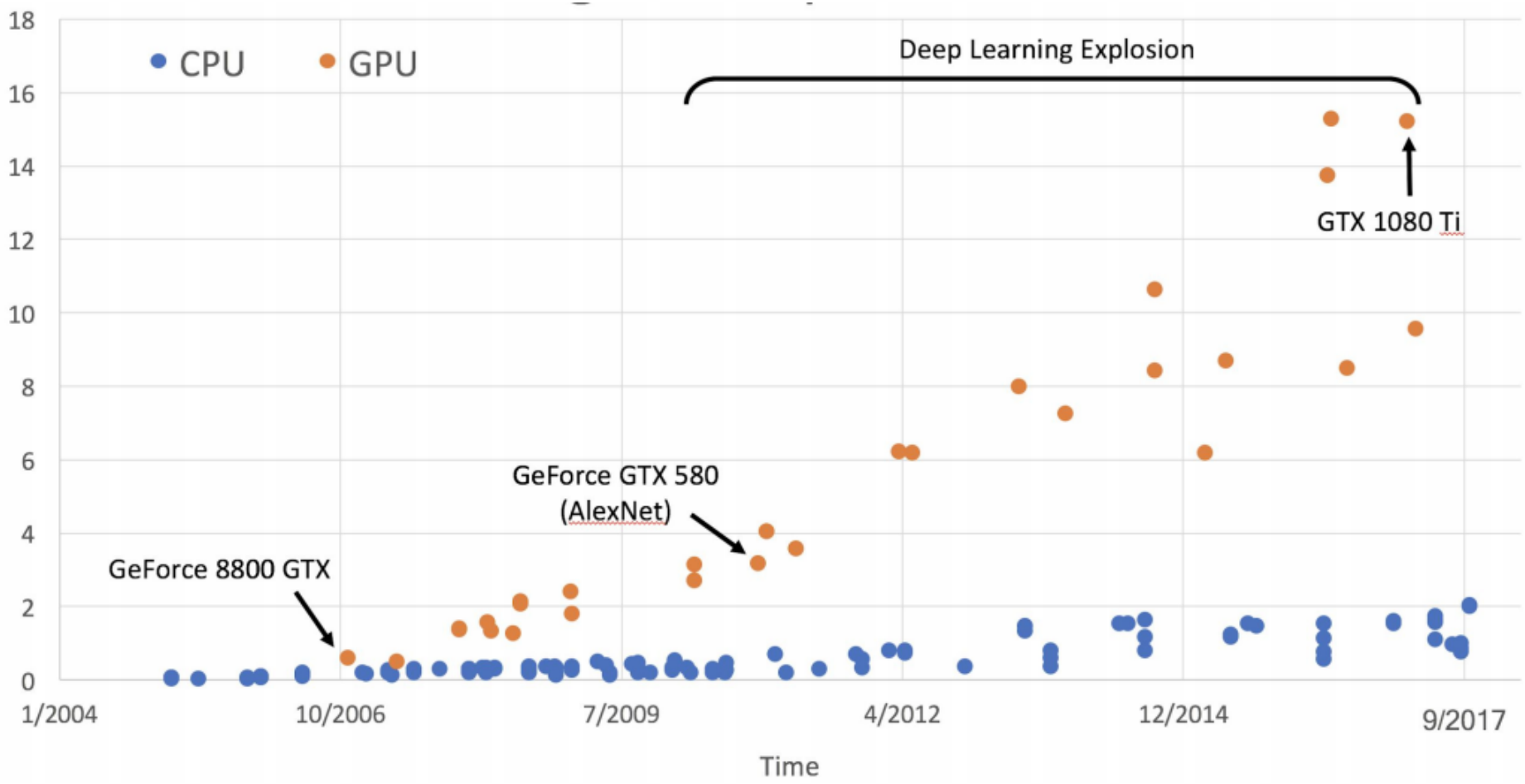
- CPU: ít nhân, nhưng mỗi nhân rất nhanh và hiệu năng cao, có khả năng xử lý thao tác phức tạp. Rất tốt cho các tác vụ tuần tự
- GPU: nhiều nhân, những mỗi nhân chậm hơn và "dốt" hơn. Rất tốt cho các tác vụ song song

Ví dụ nhân ma trận

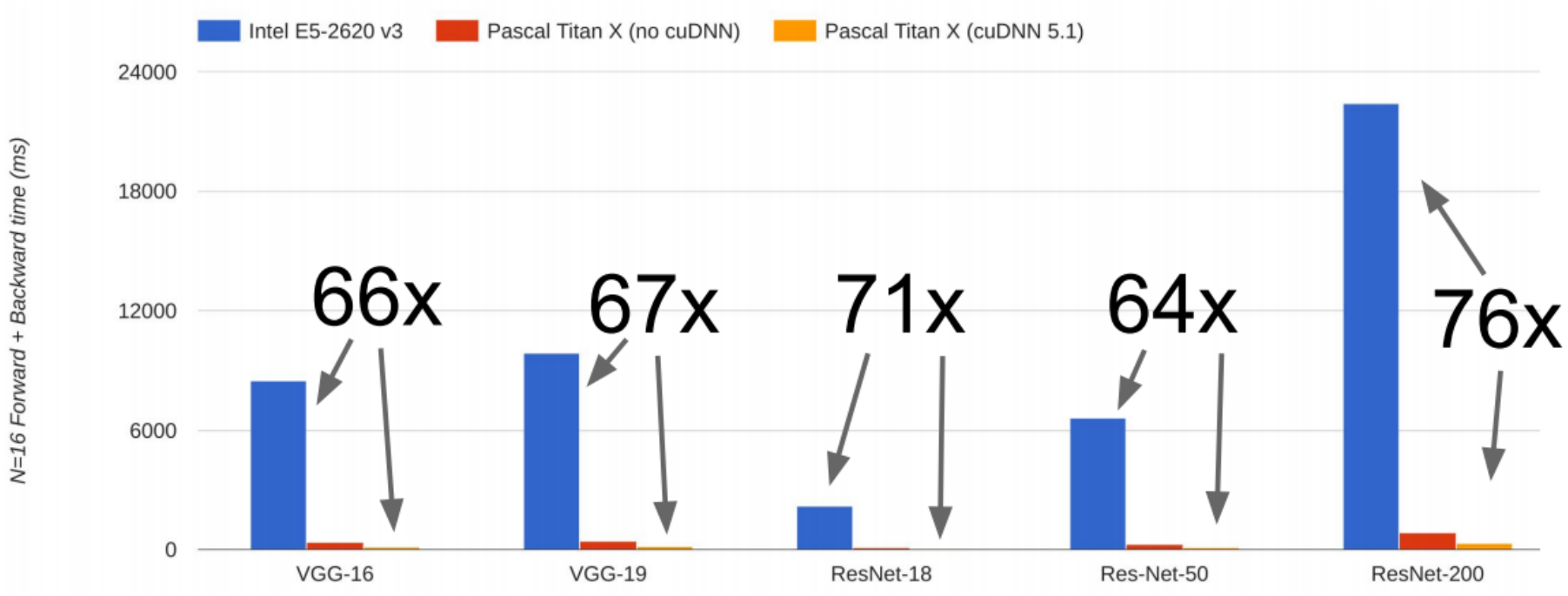


- Rất phù hợp để sử dụng GPU

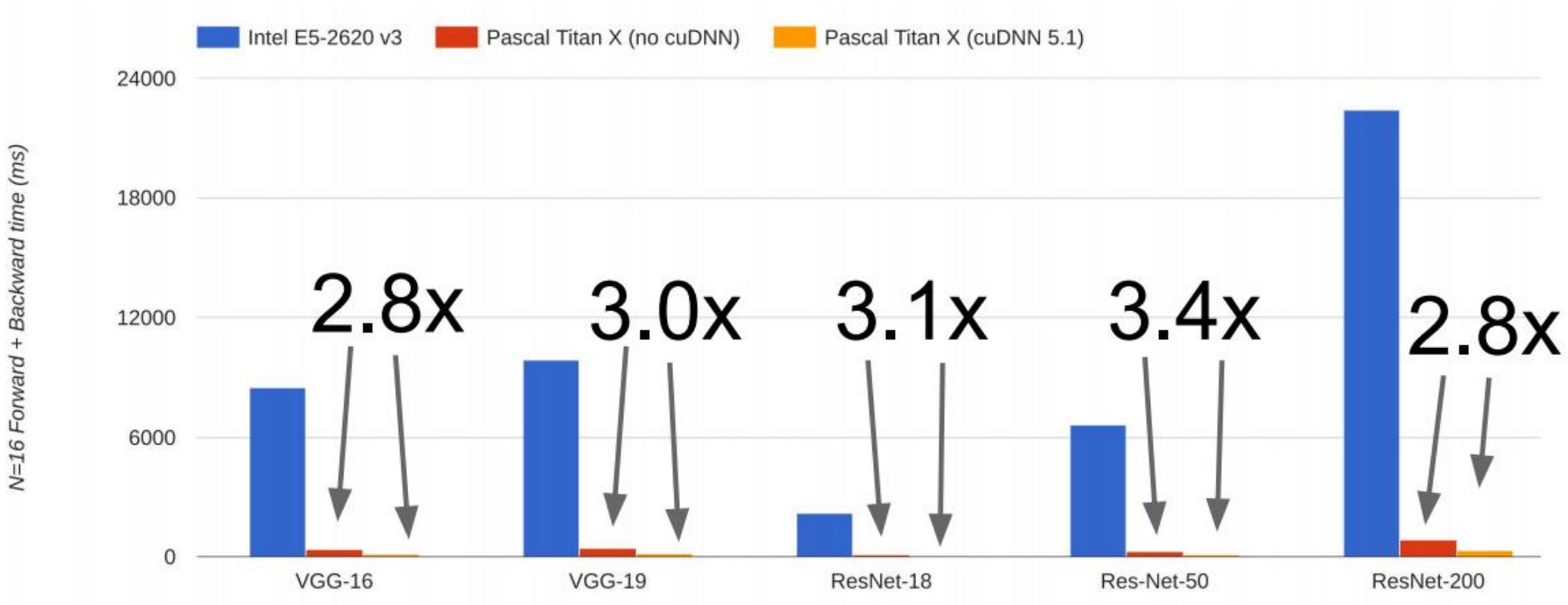
GigaFLOPs trên 1$



CPU vs GPU trong thực tế



CPU vs GPU trong thực tế

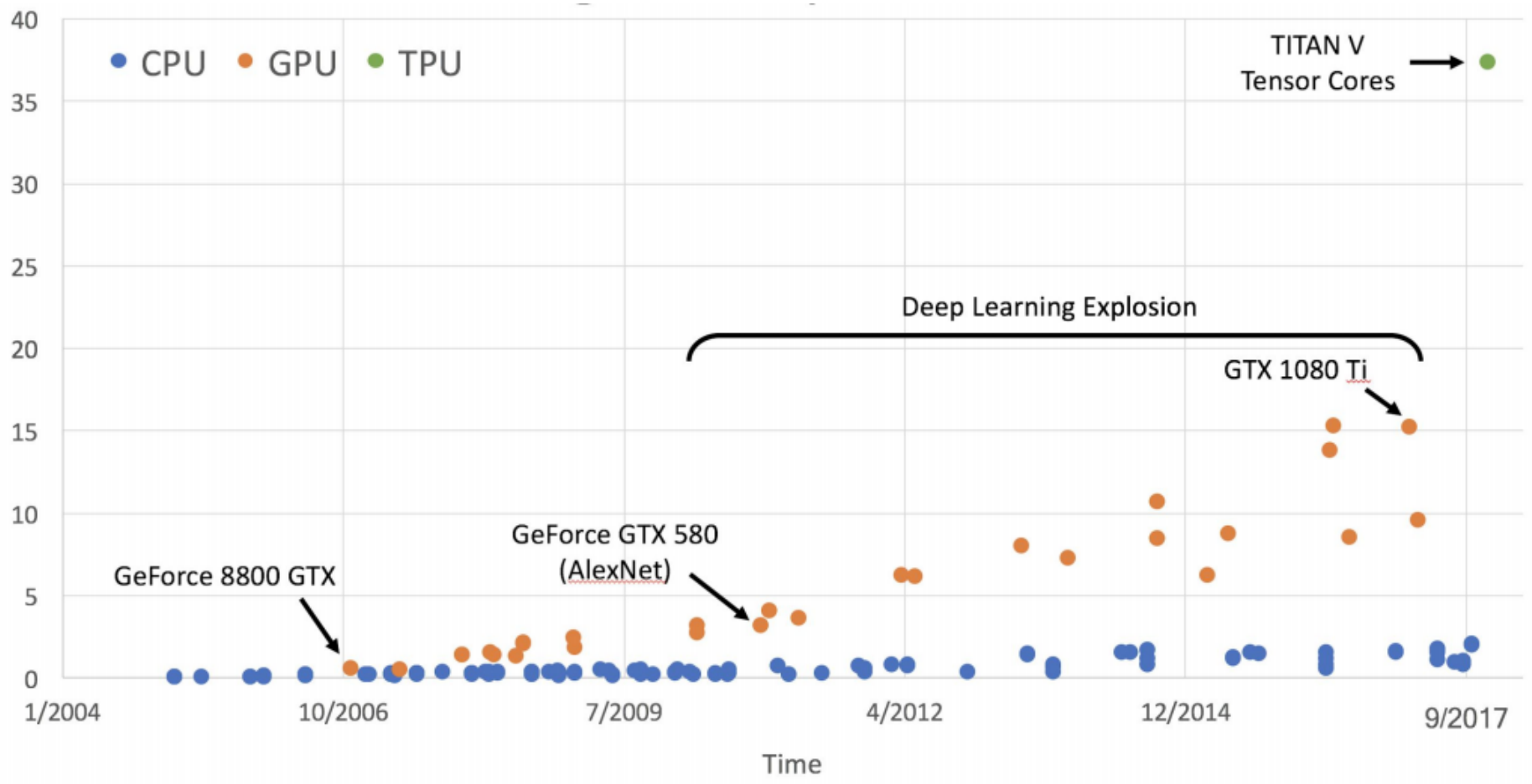


CPU vs GPU vs TPU

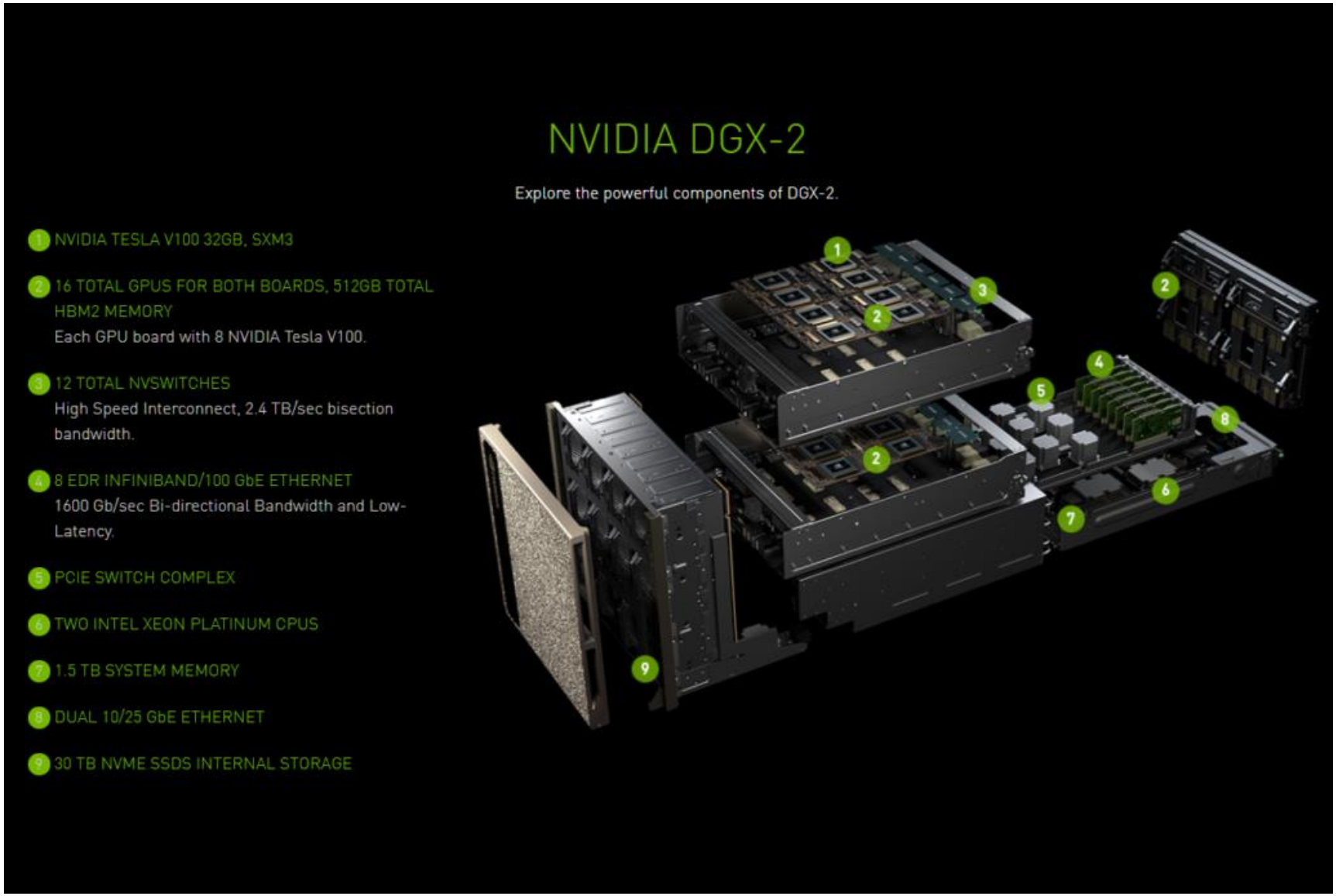
	Cores	Clock Speed	Memory	Price	Speed
CPU (Intel Core i7-7700k)	4 (8 threads with hyperthreading)	4.2 GHz	System RAM	$385	~540 GFLOPs FP32
GPU (NVIDIA RTX 2080 Ti)	3584	1.6 GHz	11 GB GDDR6	$1199	~13.4 TFLOPs FP32
TPU NVIDIA TITAN V	5120 CUDA, 640 Tensor	1.5 GHz	12GB HBM2	$2999	~14 TFLOPs FP32 ~112 TFLOP FP16
TPU Google Cloud TPU	?	?	64 GB HBM	$4.50 per hour	~180 TFLOP

- TPU: phần cứng chuyên dụng cho học sâu

GigaFLOPs trên 1$

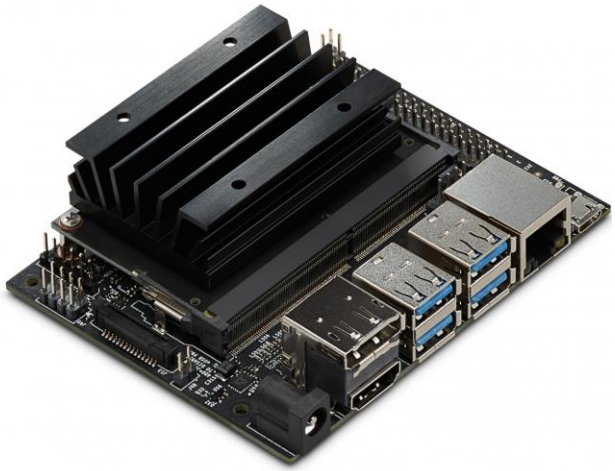


NVIDIA DGX-2

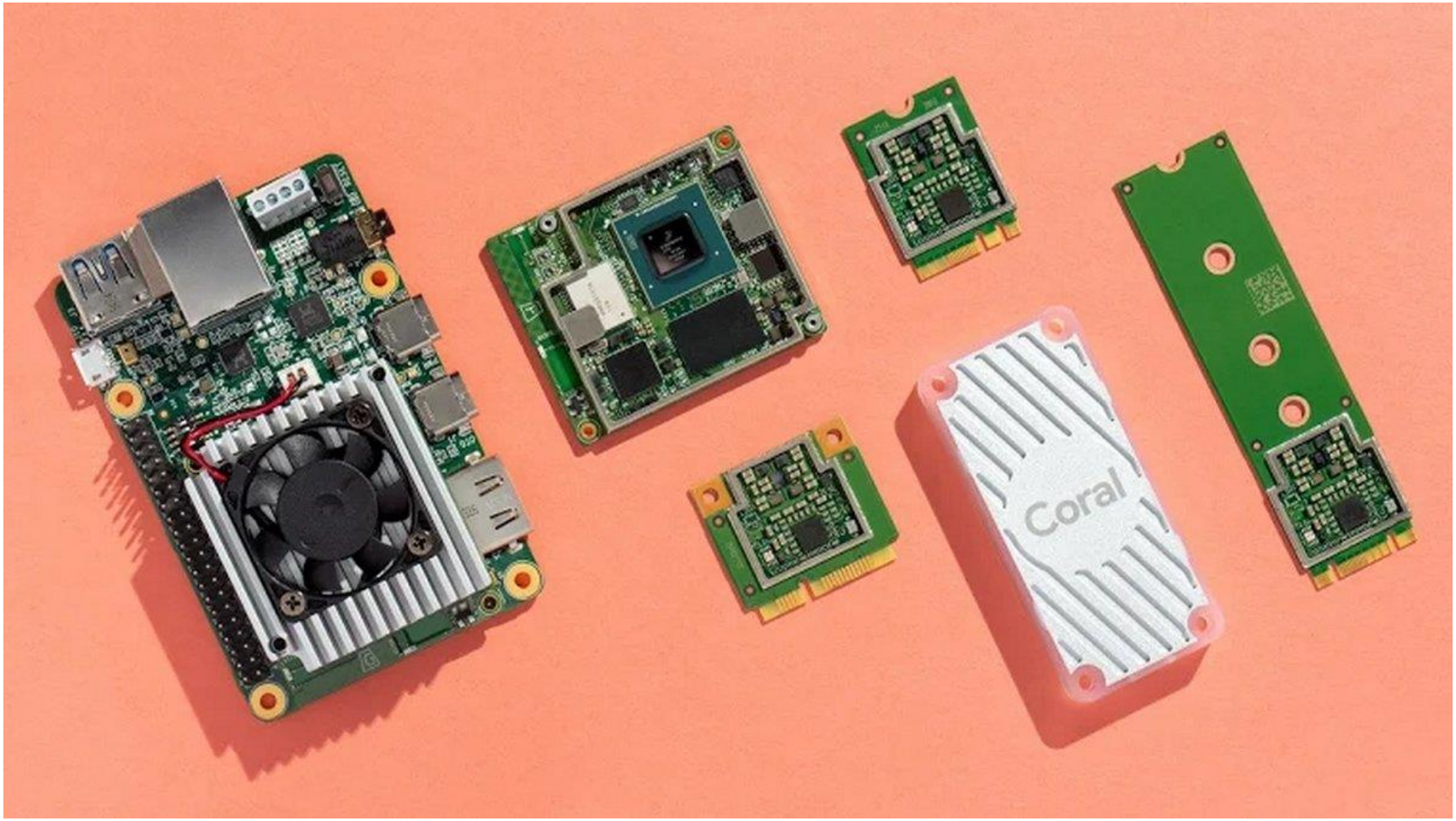

NVIDIA DGX-2
Explore the powerful components of DGX-2.
1 NVIDIA TESLA V100 32GB, SXM3
2 16 TOTAL GPUS FOR BOTH BOARDS, 512GB TOTAL HBM2 MEMORY
Each GPU board with 8 NVIDIA Tesla V100.
3 12 TOTAL NVSWITCHES
High Speed Interconnect, 2.4 TB/sec bisection bandwidth.
4 8 EDR INFINIBAND/100 GbE ETHERNET
1600 Gb/sec Bi-directional Bandwidth and Low-Latency.
5 PCIE SWITCH COMPLEX
6 TWO INTEL XEON PLATINUM CPUS
7 1.5 TB SYSTEM MEMORY
8 DUAL 10/25 GbE ETHERNET
9 30 TB NVME SSDS INTERNAL STORAGE

Thiết bị biên NVidia

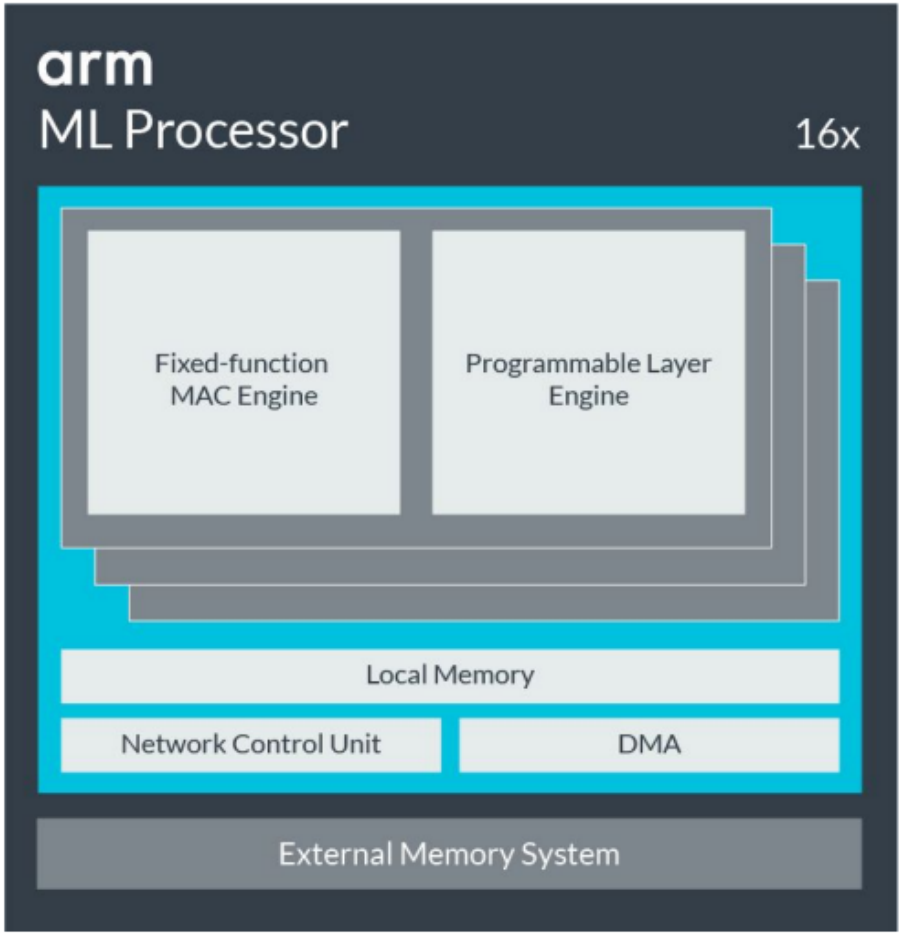
Jetson Nano	Jetson TX2 Series	Jetson Xavier NX	Jetson AGX Xavier Series
129 USD	**Starting at 249 USD**	**Starting at 399 USD**	**Starting at 599 USD**
NVIDIA Maxwell™ architecture with 128 NVIDIA CUDA® cores	NVIDIA Pascal™ architecture with 256 NVIDIA CUDA cores	NVIDIA Volta™ architecture with 384 NVIDIA CUDA cores and 48 Tensor cores	NVIDIA Volta™ architecture with up to 512 NVIDIA CUDA cores and up to 64 Tensor cores

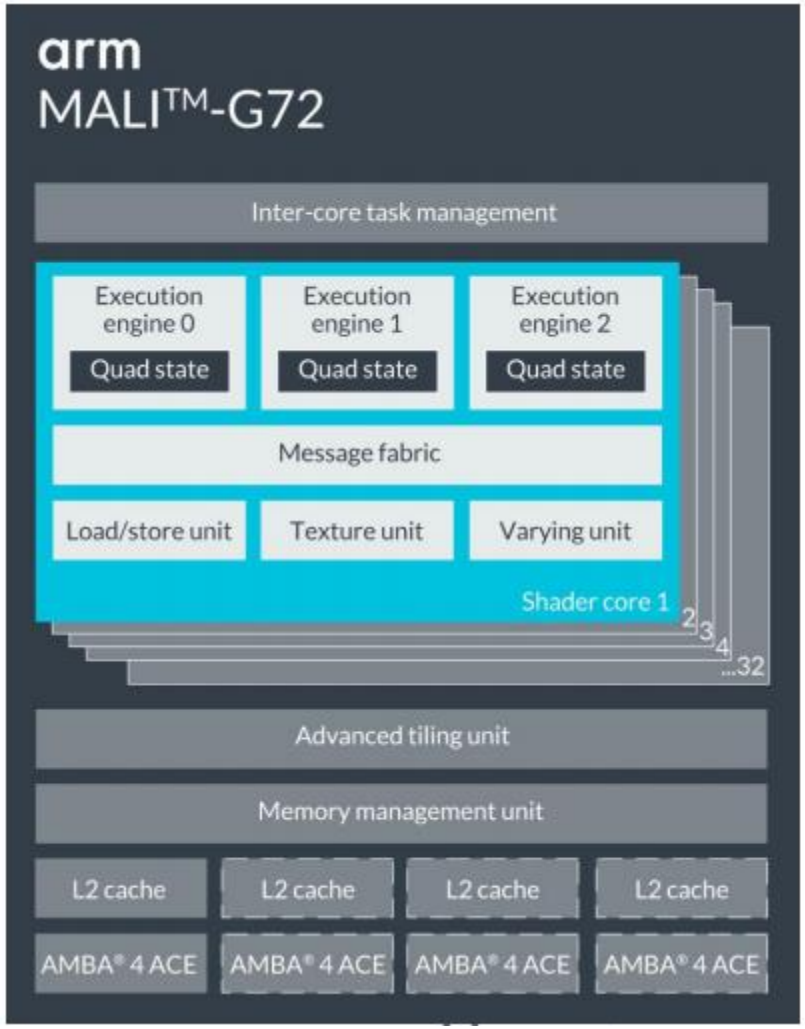


Thiết bị biên Google Coral

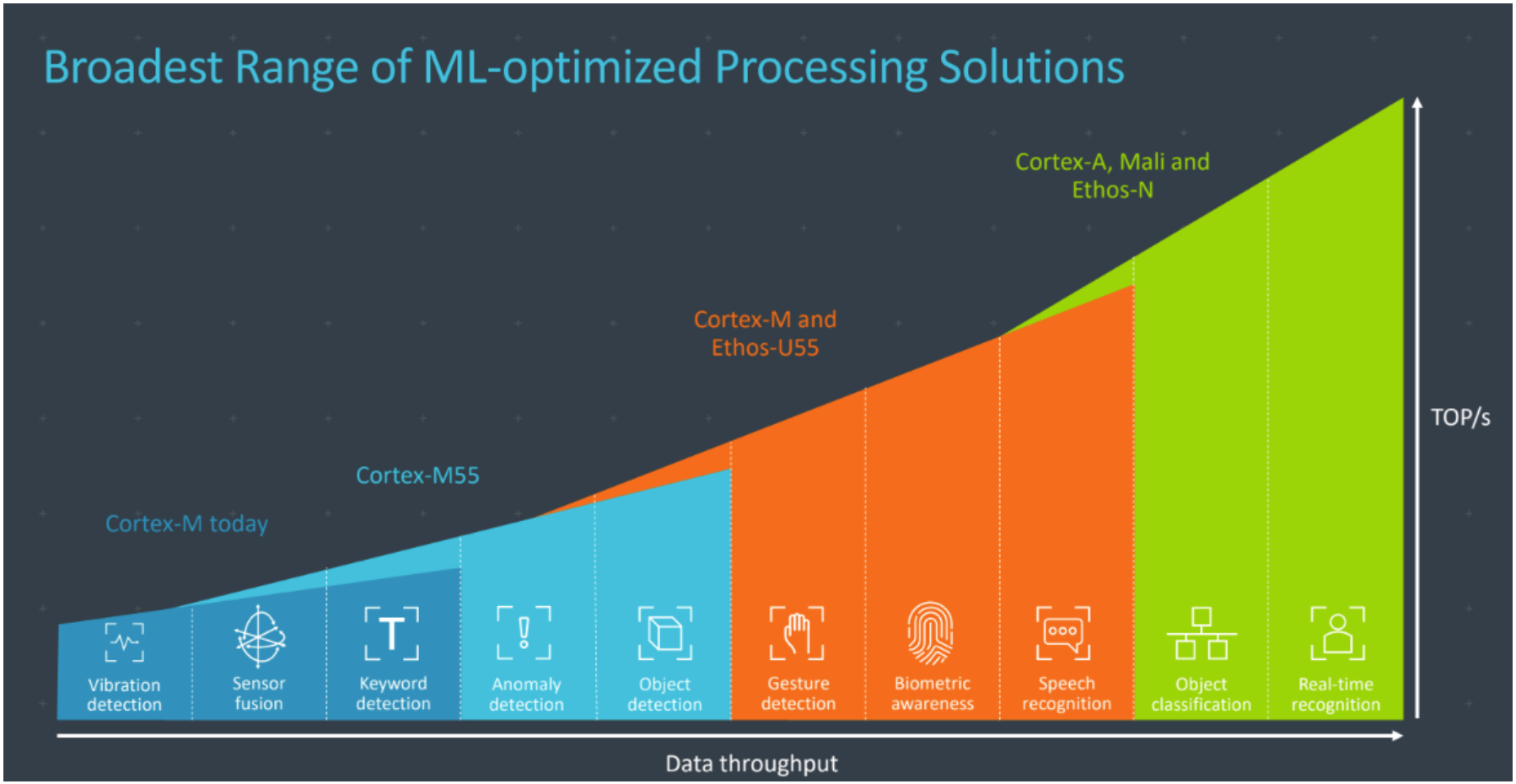


Thiết bị biên ARM



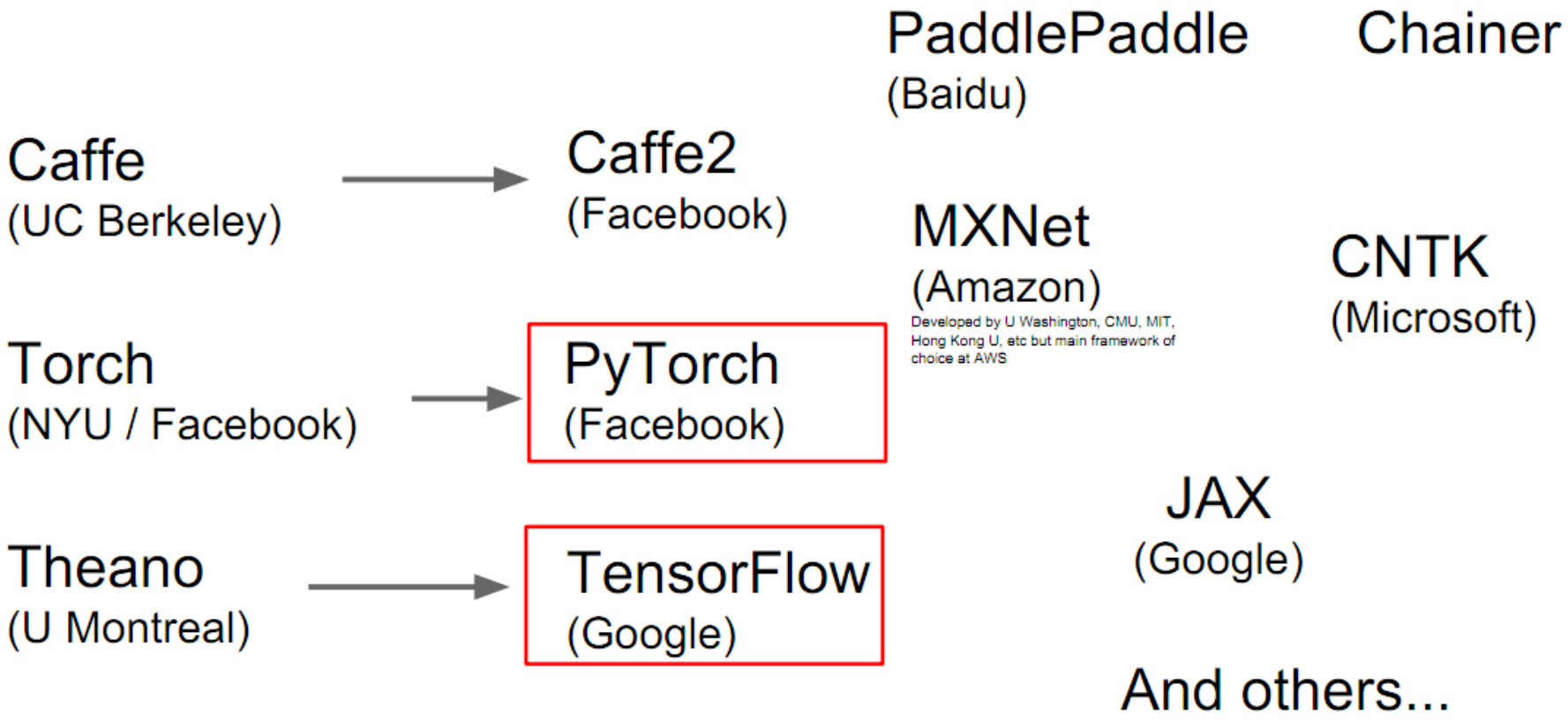


Thiết bị biên ARM NPU



Các nền tảng lập trình cho học sâu

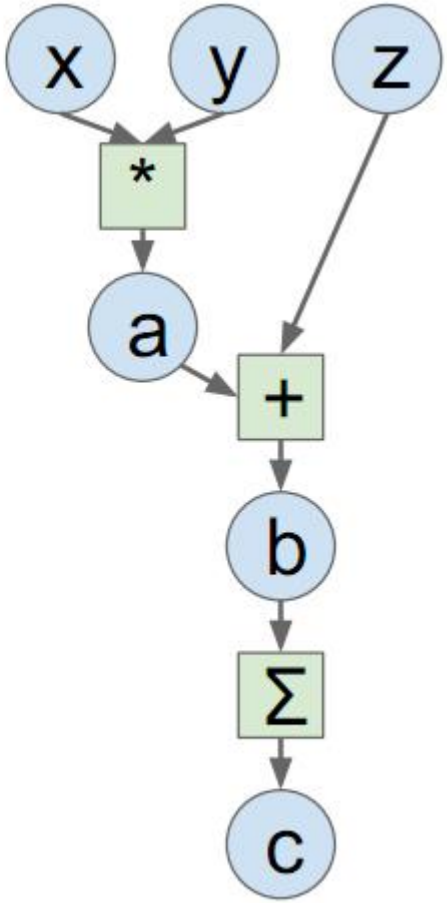
Rất nhiều nền tảng…



Đồ thị tính toán

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



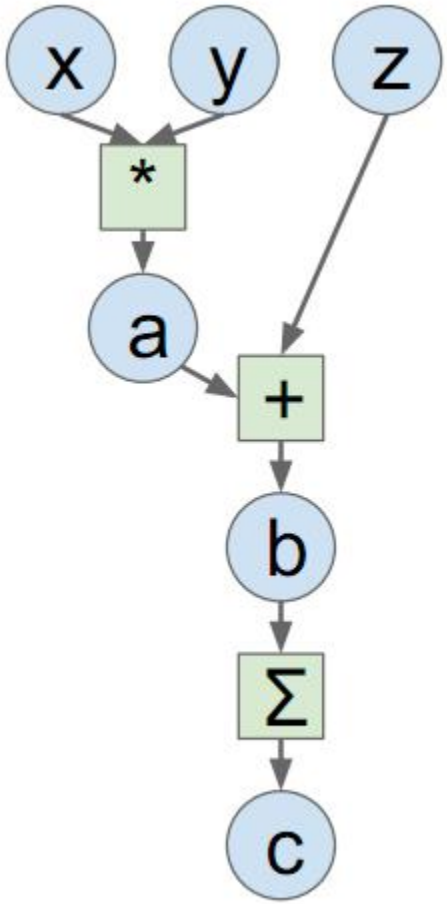
- **Ưu điểm:** API sáng sủa, dễ lập trình các tác vụ tính toán
- **Nhược điểm:** Phải tự lập trình Backprop; Không chạy được trên GPU

Đồ thị tính toán

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)
```

```
grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



PyTorch

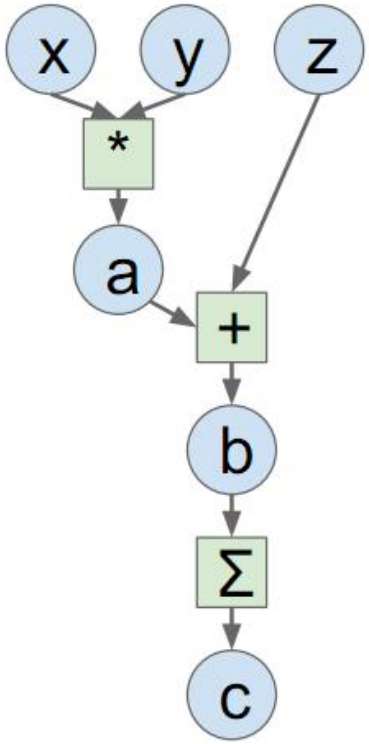
```
import torch

N, D = 3, 4
x = torch.randn(N, D)
y = torch.randn(N, D)
z = torch.randn(N, D)

a = x * y
b = a + z
c = torch.sum(b)
```

- Lập trình giống như Numpy!

Đồ thị tính toán

Numpy

```
import numpy as np
np.random.seed(0)

N, D = 3, 4

x = np.random.randn(N, D)
y = np.random.randn(N, D)
z = np.random.randn(N, D)

a = x * y
b = a + z
c = np.sum(b)

grad_c = 1.0
grad_b = grad_c * np.ones((N, D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_x = grad_a * y
grad_y = grad_a * x
```



PyTorch

```
import torch

N, D = 3, 4
x = torch.randn(N, D, requires_grad=True)
y = torch.randn(N, D)
z = torch.randn(N, D)

a = x * y
b = a + z
c = torch.sum(b)

c.backward()
print(x.grad)
```

- PyTorch tự động tính gradient

PyTorch: Tensors

- PyTorch Tensors giống numpy arrays, nhưng có thể chạy trên GPU.
- PyTorch Tensor API gần như giống hệt numpy!
- Đây là ví du huấn luyện mạng nơ-ron hai lớp sử dụng PyTorch Tensors:

```
import torch

device = torch.device('cpu')

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in, device=device)
y = torch.randn(N, D_out, device=device)
w1 = torch.randn(D_in, H, device=device)
w2 = torch.randn(H, D_out, device=device)

learning_rate = 1e-6
for t in range(500):
    h = x.mm(w1)
    h_relu = h.clamp(min=0)
    y_pred = h_relu.mm(w2)
    loss = (y_pred - y).pow(2).sum()

    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)

    w1 -= learning_rate * grad_w1
    w2 -= learning_rate * grad_w2
```

PyTorch: Autograd

- Tạo Tensors với requires_grad=True để bật tính năng autograd
- Torch.no_grad nghĩa là không đừng đưa phần này vào đồ thị tính toán
- Gán gradient bằng 0 sau mỗi bước lặp vì PyTorch tích lũy (accumulate) gradient, thuận tiện hơn khi Backprop trong mạng RNN)

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()

    with torch.no_grad():
        w1 -= learning_rate * w1.grad
        w2 -= learning_rate * w2.grad
        w1.grad.zero_()
        w2.grad.zero_()
```

PyTorch: nn

- Higher-level wrapper làm việc với mạng nơ-ron
- Giúp lập trình mọi thứ dễ dàng hơn

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-2
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    with torch.no_grad():
        for param in model.parameters():
            param -= learning_rate * param.grad
    model.zero_grad()
```

PyTorch: optim

- Có thể dùng sẵn các giải thuật tối ưu trong PyTorch, chẳng hạn như Adam

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out))

learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(),
                             lr=learning_rate)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

    loss.backward()

    optimizer.step()
    optimizer.zero_grad()
```

PyTorch: nn

- Có thể định nghĩa module mới trong PyTorch
- Module có thể chứa trọng số hoặc các module khác
- PyTorch tự động xử lý Autograd cho module mới

```
import torch

class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

model = TwoLayerNet(D_in, H, D_out)

optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    y_pred = model(x)
    loss = torch.nn.functional.mse_loss(y_pred, y)

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

PyTorch: Pretrained models

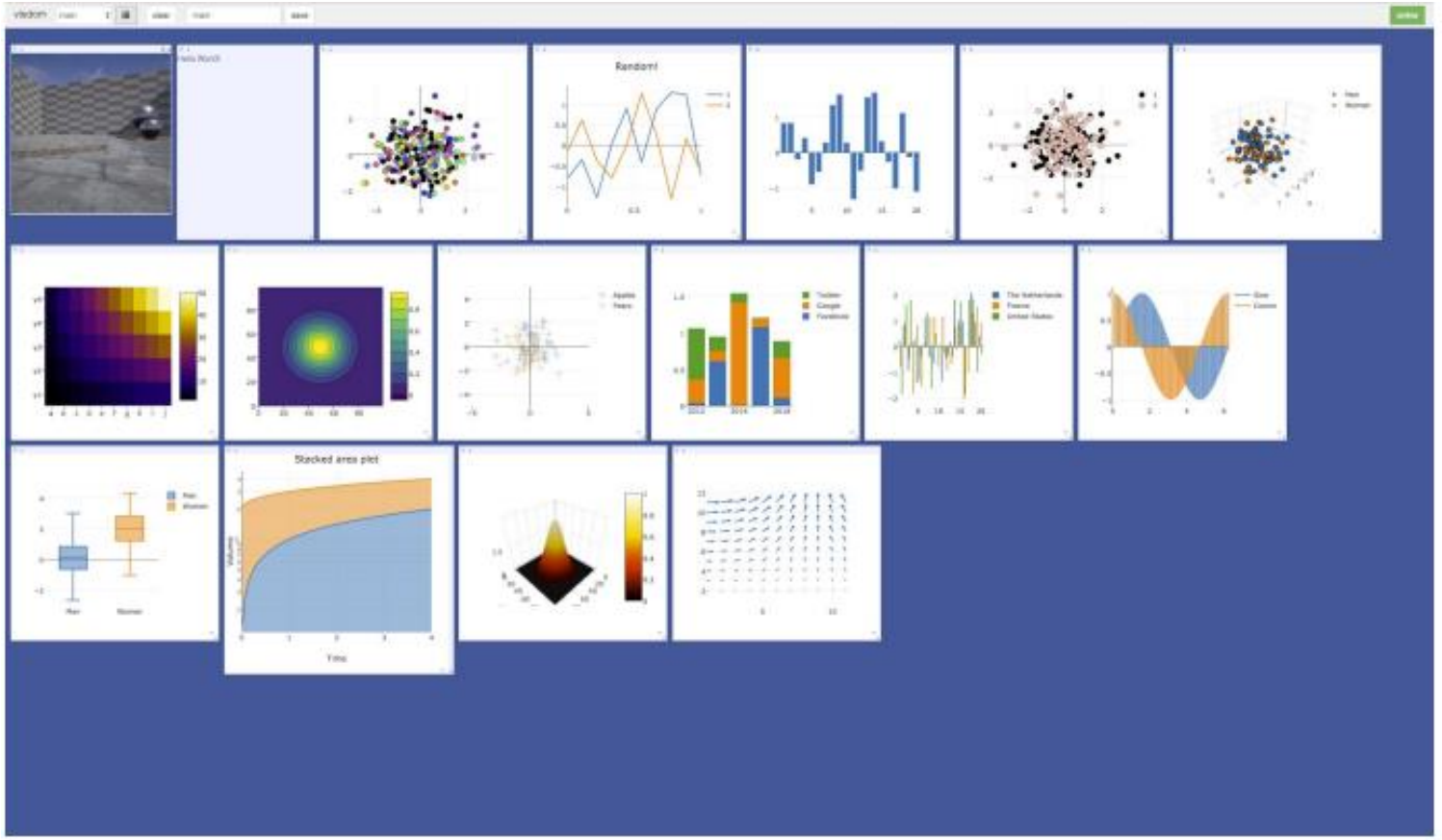
- PyTorch có sẵn một số mô hình được huấn luyện sẵn. Có thể dùng trực tiếp các mô hình này.

Super easy to use pretrained models with torchvision
https://github.com/pytorch/vision

```
import torch
import torchvision

alexnet = torchvision.models.alexnet(pretrained=True)
vgg16 = torchvision.models.vgg16(pretrained=True)
resnet101 = torchvision.models.resnet101(pretrained=True)
```

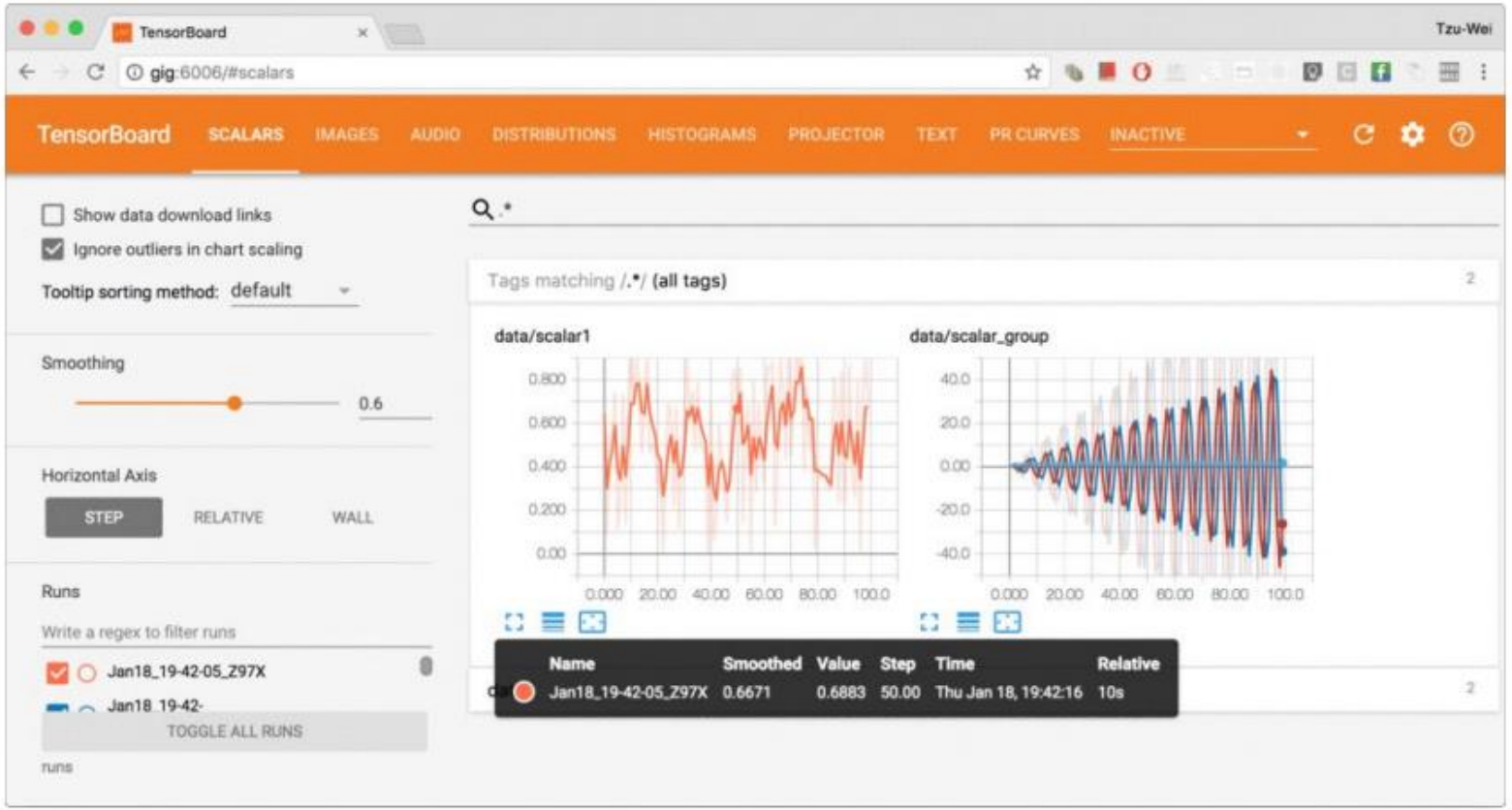
PyTorch: Visdom

- Công cụ hỗ trợ trực quan hóa quá trình tính toán
- Hiện chưa hỗ trợ trực quan cấu trúc đồ thị tính toán



PyTorch: tensorboardx

- Công cụ trực quan hóa phát triển dựa trên Tensorboard của Tensorflow
- pip install tensorboardx
- https://github.com/lanpa/tensorboardX



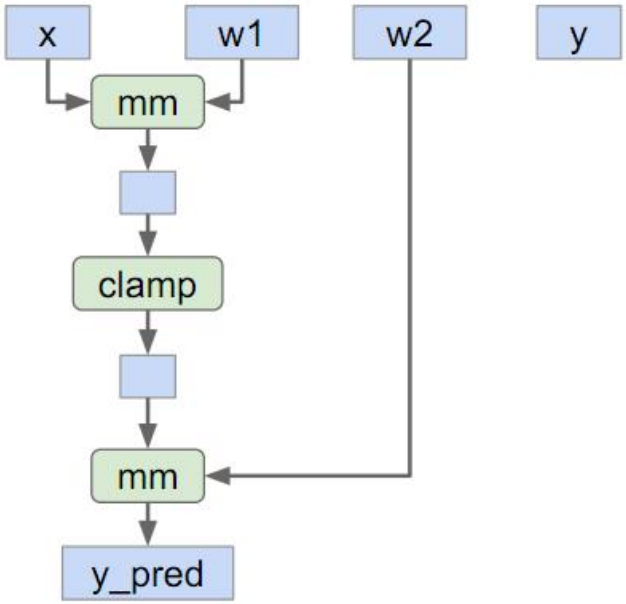
PyTorch: Đồ thị tính toán động

- Tạo tensor


```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Đồ thị tính toán động

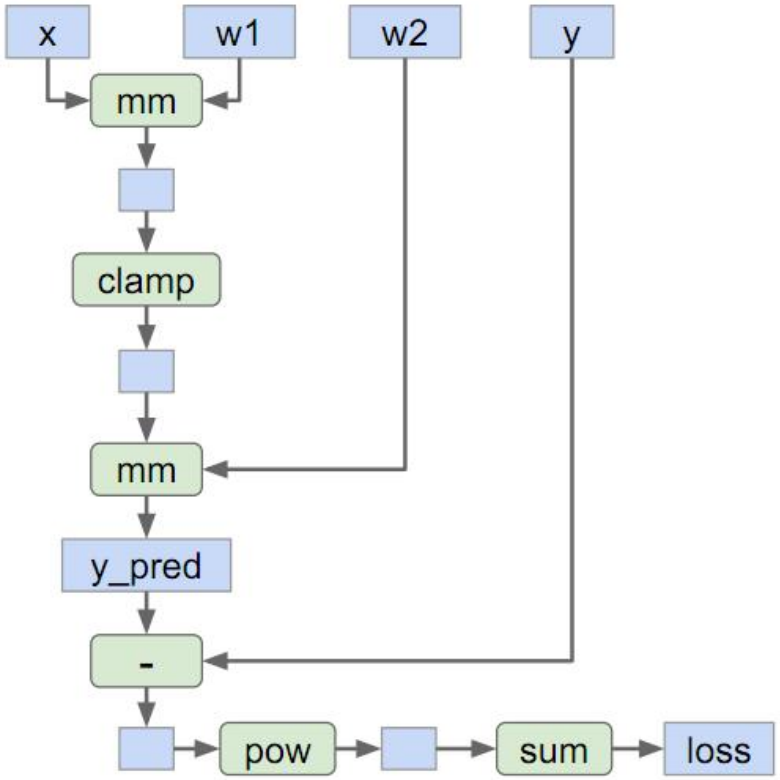
- Xây dựng đồ thị và thực hiện tính toán




```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Đồ thị tính toán động

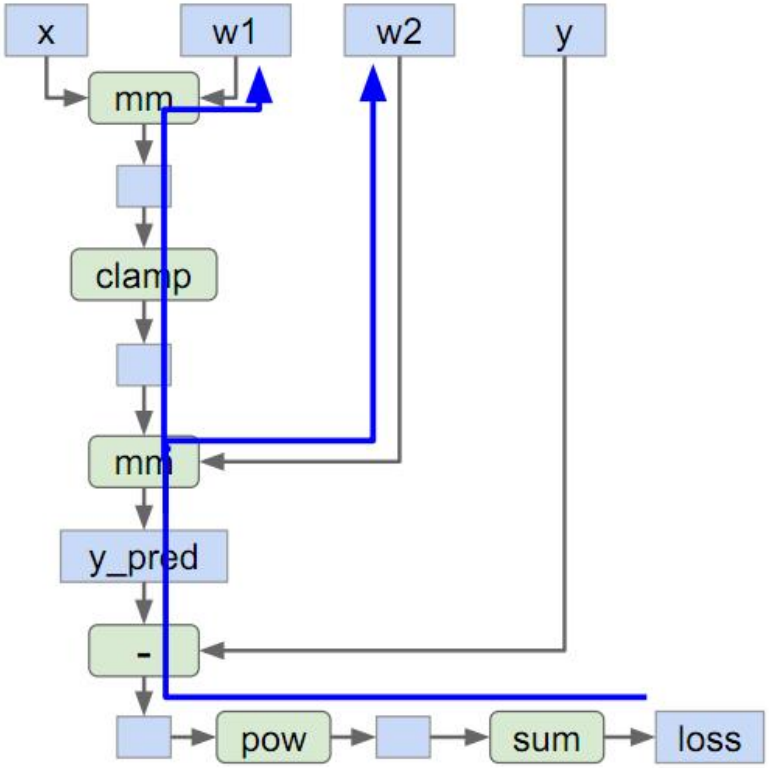
- Xây dựng đồ thị và thực hiện tính toán




```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Đồ thị tính toán động

- Tìm kiếm đường đi trên đồ thị từ hàm mục tiêu tới w1 và w2, sau đó thực hiện tính toán




```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Đồ thị tính toán động

- Đến bước lặp tiếp theo xóa tất cả đồ thị và đường lan truyền ngược ở bước trước, xây dựng tất cả lại từ đầu
- Dường như không hiệu quả, đặc biệt khi xây dựng cùng một đồ thị nhiều lần...

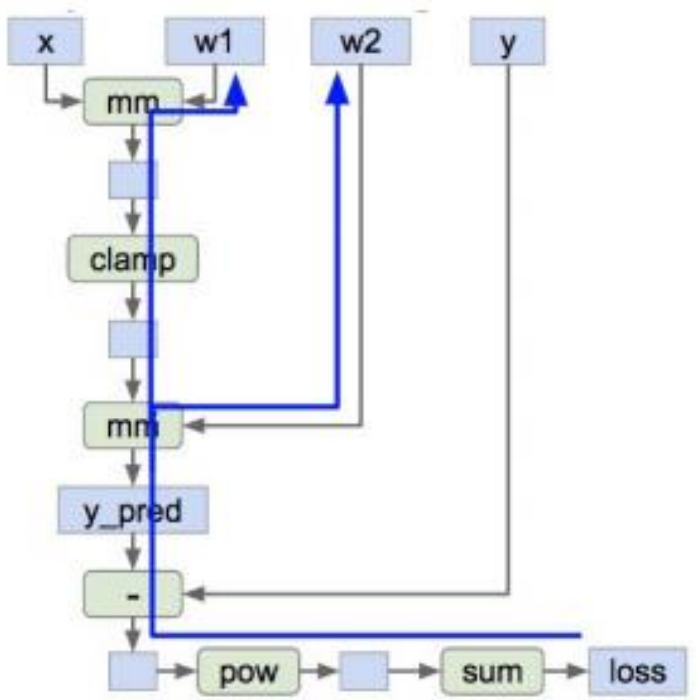

```
import torch

N, D_in, H, D_out = 64, 1000, 100, 10
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)
w1 = torch.randn(D_in, H, requires_grad=True)
w2 = torch.randn(H, D_out, requires_grad=True)

learning_rate = 1e-6
for t in range(500):
    y_pred = x.mm(w1).clamp(min=0).mm(w2)
    loss = (y_pred - y).pow(2).sum()

    loss.backward()
```

PyTorch: Đồ thị tính toán tĩnh

Static graph

Bước 1: Xây dựng đồ thị tính toán

Bước 2: Dùng đồ thị này để thực hiện tính toán cho tất cả các bước lặp




```
graph = build_graph()

for x_batch, y_batch in loader:
    run_graph(graph, x=x_batch, y=y_batch)
```

Tensorflow Pre2.0

- Bước 1: Xây dựng đồ thị tính toán
- Bước 2: Chạy đồ thị tính toán này nhiều lần

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 2.0

- Chế độ Eager Execution (thực thi nóng) của TensorFlow là môi trường lập trình mệnh lệnh cho phép thực thi các phép toán ngay tức thời mà không cần xây dựng đồ thị tính toán: các phép toán trả về các giá trị cụ thể thay vì xây dựng đồ thị tính toán rồi chạy sau.
- Điều này giúp chúng ta dễ dàng bắt đầu với các mô hình TensorFlow hơn và dễ gỡ lỗi hơn.

Tensorflow 2.0 vs Pre2.0

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0:
"Eager" Mode by default

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 1.13

Tensorflow 2.0 vs Pre2.0

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0:
"Eager" Mode by default

```
N, D, H = 64, 1000, 100
x = tf.placeholder(tf.float32, shape=(N, D))
y = tf.placeholder(tf.float32, shape=(N, D))
w1 = tf.placeholder(tf.float32, shape=(D, H))
w2 = tf.placeholder(tf.float32, shape=(H, D))

h = tf.maximum(tf.matmul(x, w1), 0)
y_pred = tf.matmul(h, w2)
diff = y_pred - y
loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))

grad_w1, grad_w2 = tf.gradients(loss, [w1, w2])

with tf.Session() as sess:
    values = {x: np.random.randn(N, D),
              w1: np.random.randn(D, H),
              w2: np.random.randn(H, D),
              y: np.random.randn(N, D),}
    out = sess.run([loss, grad_w1, grad_w2],
                   feed_dict=values)
    loss_val, grad_w1_val, grad_w2_val = out
```

Tensorflow 1.13

Tensorflow 2.0: Neural Network

- Biến mảng numpy thành TF tensor

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

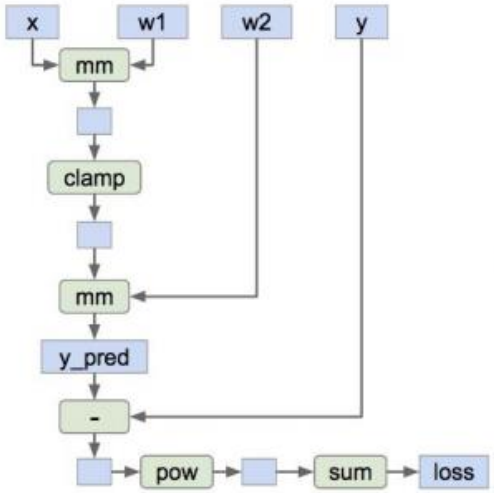
Tensorflow 2.0: Neural Network

- Sử dụng tf.GradientTape() để xây dựng đồ thị tính toán động

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0: Neural Network

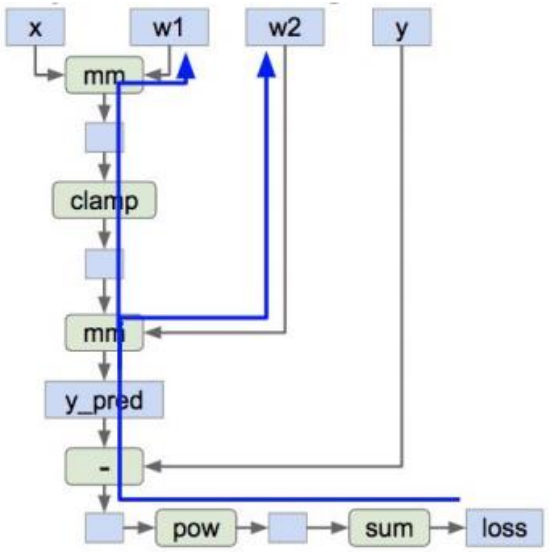
- Tất cả các phép toán trong bước forward được lưu vết để phục vụ việc tính toán gradient về sau.




```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0: Neural Network

- tape.gradient() sử dụng đồ thị tính toán đã được lưu vết trước đó để tính toán gradient




```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

with tf.GradientTape() as tape:
  h = tf.maximum(tf.matmul(x, w1), 0)
  y_pred = tf.matmul(h, w2)
  diff = y_pred - y
  loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
gradients = tape.gradient(loss, [w1, w2])
```

Tensorflow 2.0: Neural Network

- Huấn luyện mạng nơ-ron: chạy đồ thị tính toán nhiều bước lặp, sử dụng gradient để cập nhật các trọng số

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights
```

```
learning_rate = 1e-6
for t in range(50):
  with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
  gradients = tape.gradient(loss, [w1, w2])
  w1.assign(w1 - learning_rate * gradients[0])
  w2.assign(w2 - learning_rate * gradients[1])
```

Tensorflow 2.0: Neural Network

- Có thể dùng giải thuật tối ưu có sẵn (optimizer) để tính gradient và cập nhật trọng số

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

optimizer = tf.optimizers.SGD(1e-6)

learning_rate = 1e-6
for t in range(50):
  with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.reduce_mean(tf.reduce_sum(diff ** 2, axis=1))
  gradients = tape.gradient(loss, [w1, w2])
  optimizer.apply_gradients(zip(gradients, [w1, w2]))
```

Tensorflow 2.0: Neural Network

- Có thể sử dụng hàm mục tiêu được định nghĩa sẵn

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
w1 = tf.Variable(tf.random.uniform((D, H)))  # weights
w2 = tf.Variable(tf.random.uniform((H, D)))  # weights

optimizer = tf.optimizers.SGD(1e-6)

for t in range(50):
  with tf.GradientTape() as tape:
    h = tf.maximum(tf.matmul(x, w1), 0)
    y_pred = tf.matmul(h, w2)
    diff = y_pred - y
    loss = tf.losses.MeanSquaredError()(y_pred, y)
  gradients = tape.gradient(loss, [w1, w2])
  optimizer.apply_gradients(zip(gradients, [w1, w2]))
```

Keras: High-Level wrapper

- Keras là lớp bao (wrapper) được xây dựng phía trên tensorflow

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

losses = []
for t in range(50):
  with tf.GradientTape() as tape:
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
  gradients = tape.gradient(
      loss, model.trainable_variables)
  optimizer.apply_gradients(
      zip(gradients, model.trainable_variables))
```

Keras: High-Level wrapper

Định nghĩa mô hình như một dãy các lớp

Lấy output bằng cách gọi mô hình

Tính gradient đối với tất cả các trọng số có thể cập nhật (trainable) của mô hình

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

losses = []
for t in range(50):
  with tf.GradientTape() as tape:
    y_pred = model(x)
    loss = tf.losses.MeanSquaredError()(y_pred, y)
  gradients = tape.gradient(
      loss, model.trainable_variables)
  optimizer.apply_gradients(
      zip(gradients, model.trainable_variables))
```

Keras có thể xử lý vòng lặp cho ta

```
N, D, H = 64, 1000, 100

x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)
model.compile(loss=tf.keras.losses.MeanSquaredError(),
              optimizer=optimizer)

history = model.fit(x, y, epochs=50, batch_size=N)
```

Tensorflow 2.0: @tf.function

- @tf.function để biên dịch đồ thị tính toán tĩnh
- Đồ thị tính toán tĩnh thường nhanh hơn đồ thị tính toán động

```
N, D, H = 64, 1000, 100
x = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
y = tf.convert_to_tensor(np.random.randn(N, D), np.float32)
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(H, input_shape=(D,),
                                activation=tf.nn.relu))
model.add(tf.keras.layers.Dense(D))
optimizer = tf.optimizers.SGD(1e-1)

@tf.function
def model_static(x, y):
  y_pred = model(x)
  loss = tf.losses.MeanSquaredError()(y_pred, y)
  return y_pred, loss

def model_dynamic(x, y):
  y_pred = model(x)
  loss = tf.losses.MeanSquaredError()(y_pred, y)
  return y_pred, loss

print("static graph:",
      timeit.timeit(lambda: model_static(x, y), number=10))
print("dynamic graph:",
      timeit.timeit(lambda: model_dynamic(x, y), number=10))
```

```
static graph: 0.14495624600000667
dynamic graph: 0.029459196999994 22
```

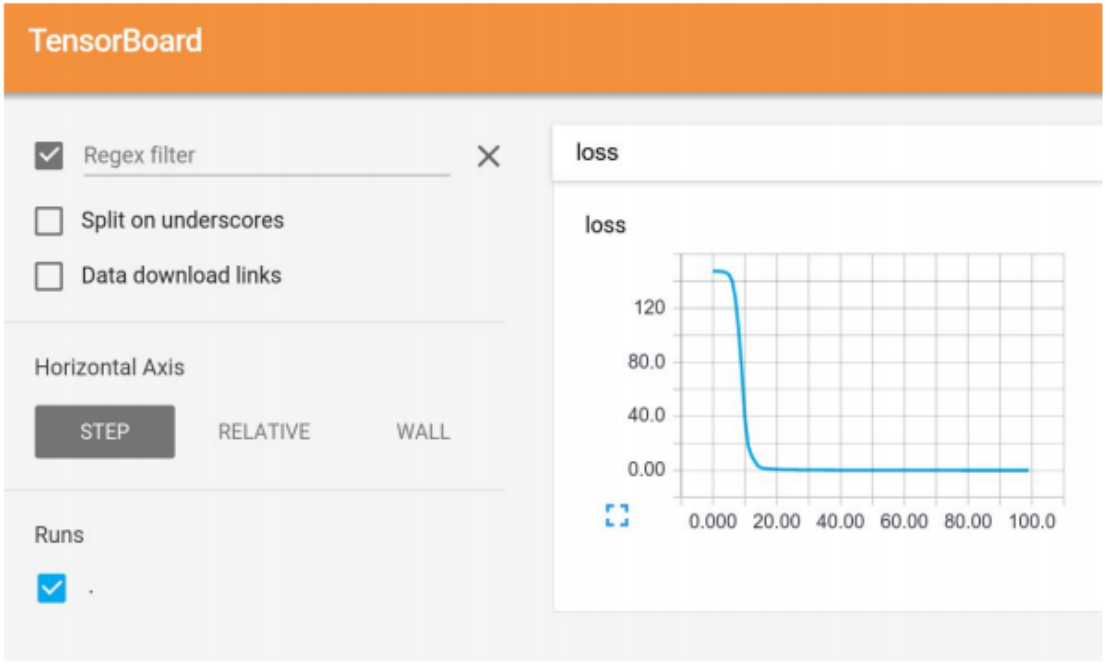
TensorFlow: Pretrained Models

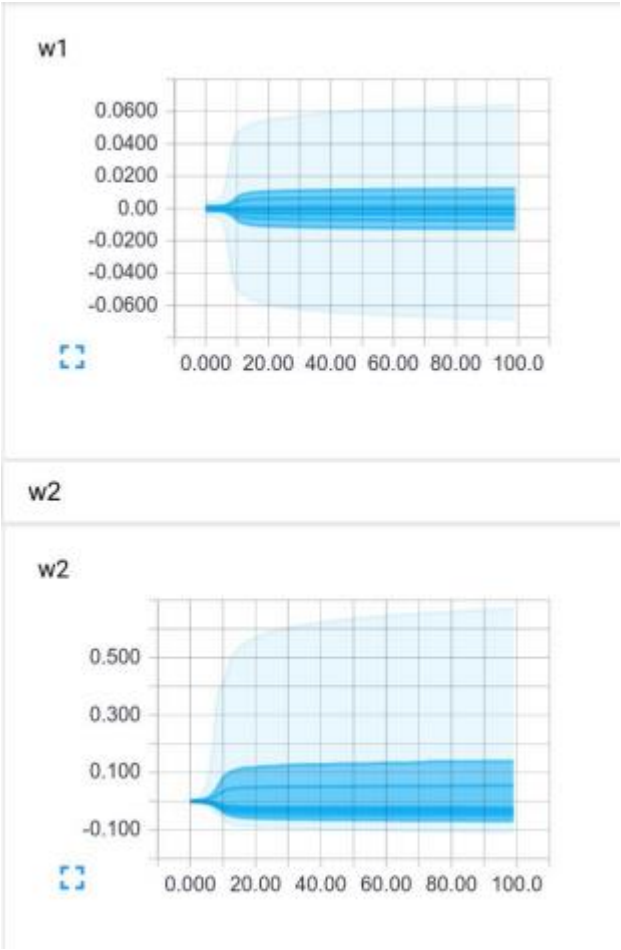
- tf.keras: https://www.tensorflow.org/api_docs/python/tf/keras/applications

- TF-Slim: https://github.com/tensorflow/models/tree/master/research/slim

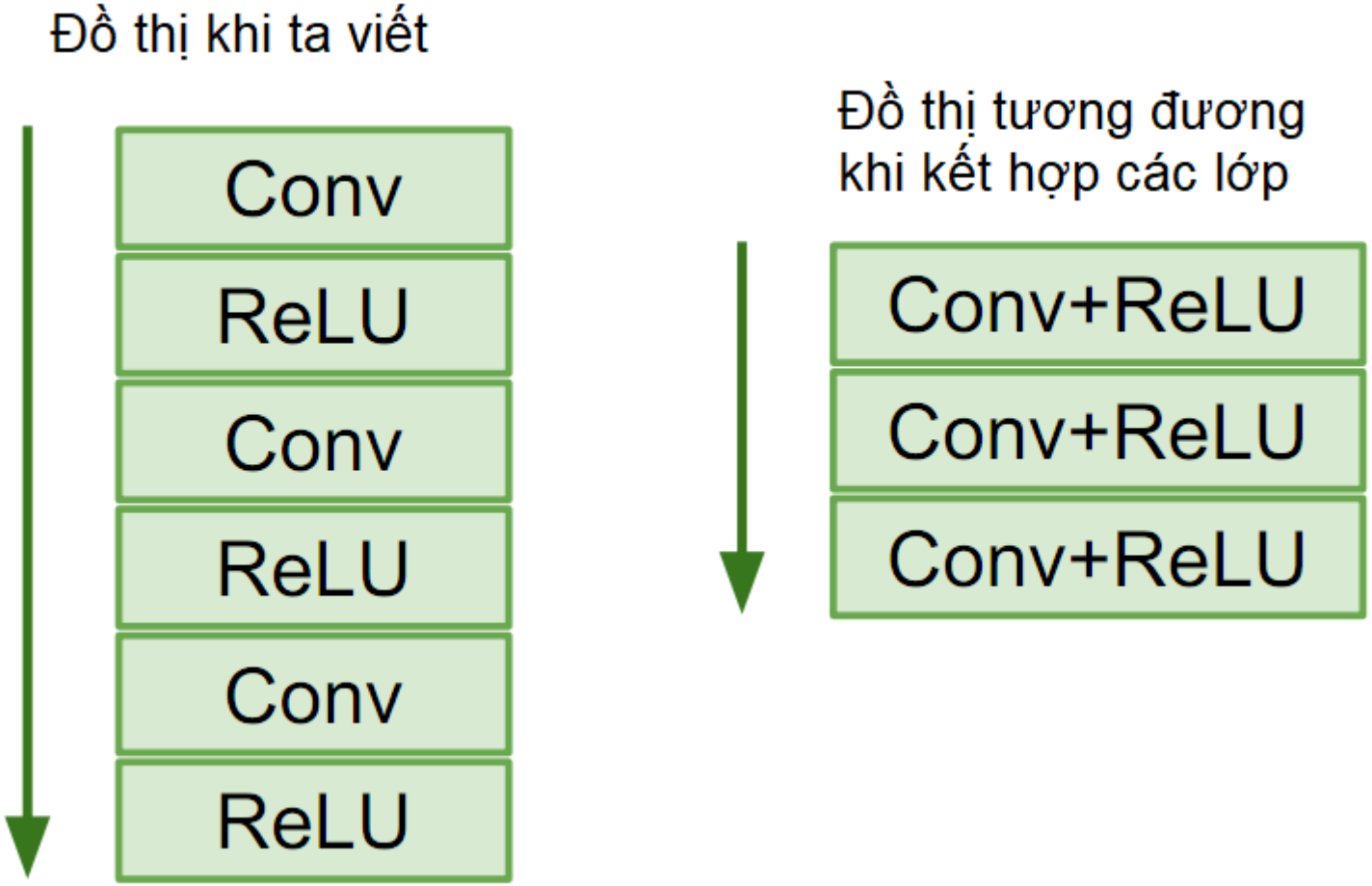
TensorFlow: Tensorboard

- Thêm log vào trong code để quan sát hàm mục tiêu, các tham số…
- Chạy server tensorboard và xem kết quả





Đồ thị tĩnh vs Đồ thị động



- Với đồ thị tĩnh, framework có thể tối ưu đồ thị cho chúng ta trước khi chạy nó

Đồ thị tĩnh vs Đồ thị động

- **Đồ thị tĩnh:**

  Một khi đã xây dựng xong có thể sử dụng tiếp và chạy nó mà không cần đoạn code xây dựng đồ thị nữa.

- **Đồ thị động:**

  Đồ thị được xây dựng và thực hiện tính toán đan xen nhau. Vì vậy luôn phải cần code để xây dựng đồ thị.

PyTorch
Dynamic Graphs

TensorFlow
Pre-2.0: Default
Static Graph
2.0+: Default
Dynamic Graph

Static PyTorch

- **Caffe2:**

https://caffe2.ai/

- **ONNX:**

https://github.com/onnx/onnx

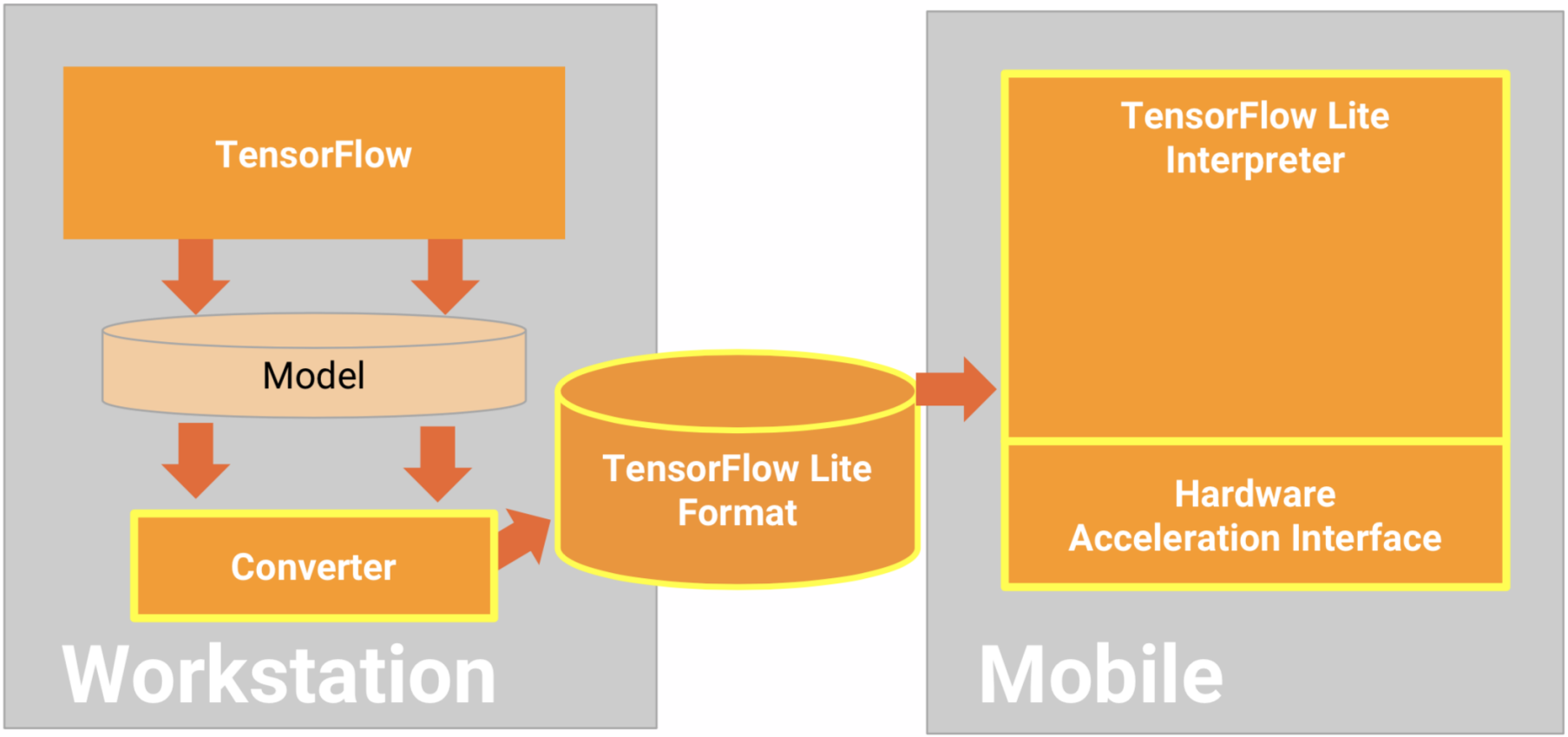
PyTorch
Dynamic Graphs
Static: ONNX, Caffe2

TensorFlow
Dynamic: Eager
Static: @tf.function

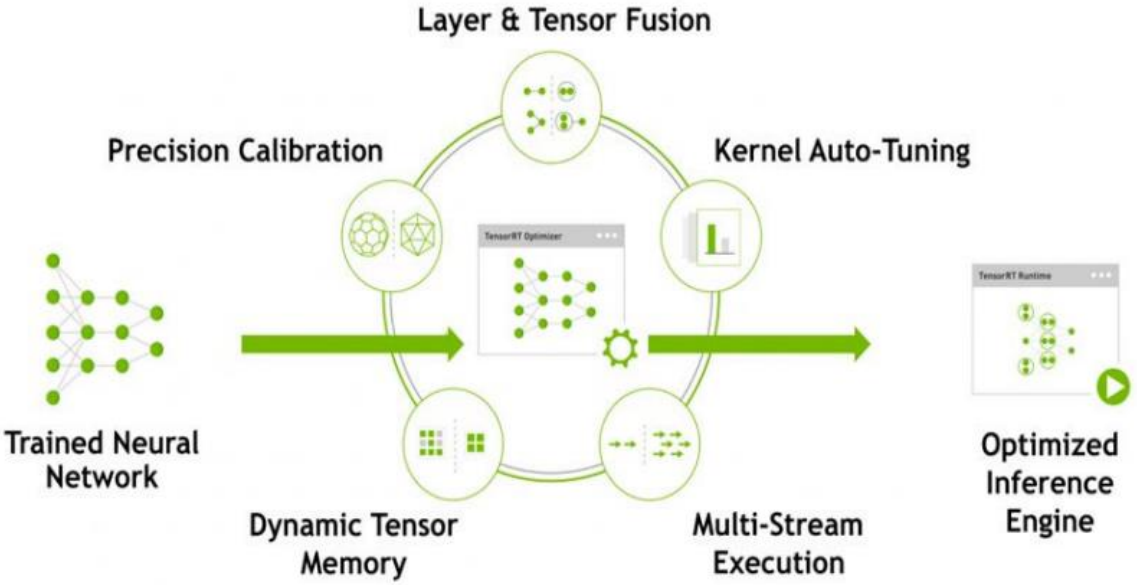
Công cụ tăng tốc và nén mạng

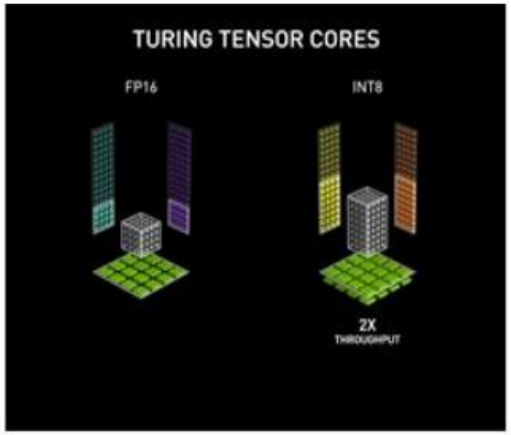
Tensorflow Lite

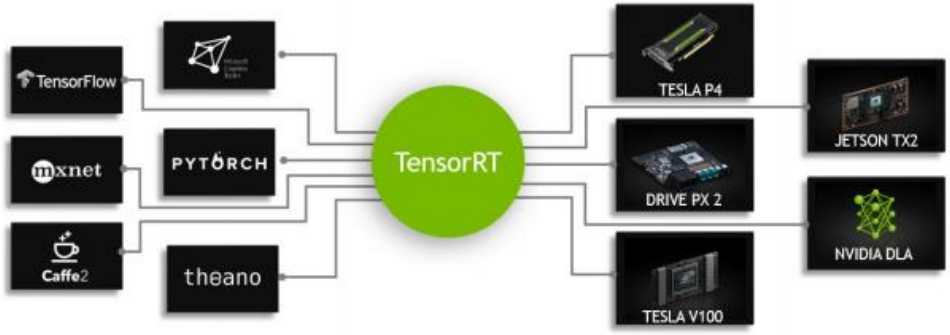
- Tensorflow Lite là một tập hợp các công cụ giúp tối ưu mô hình Tensorflow, làm mô hình nhỏ gọn hơn và suy diễn nhanh hơn trên các nền tảng di động



NVIDIA TensorRT

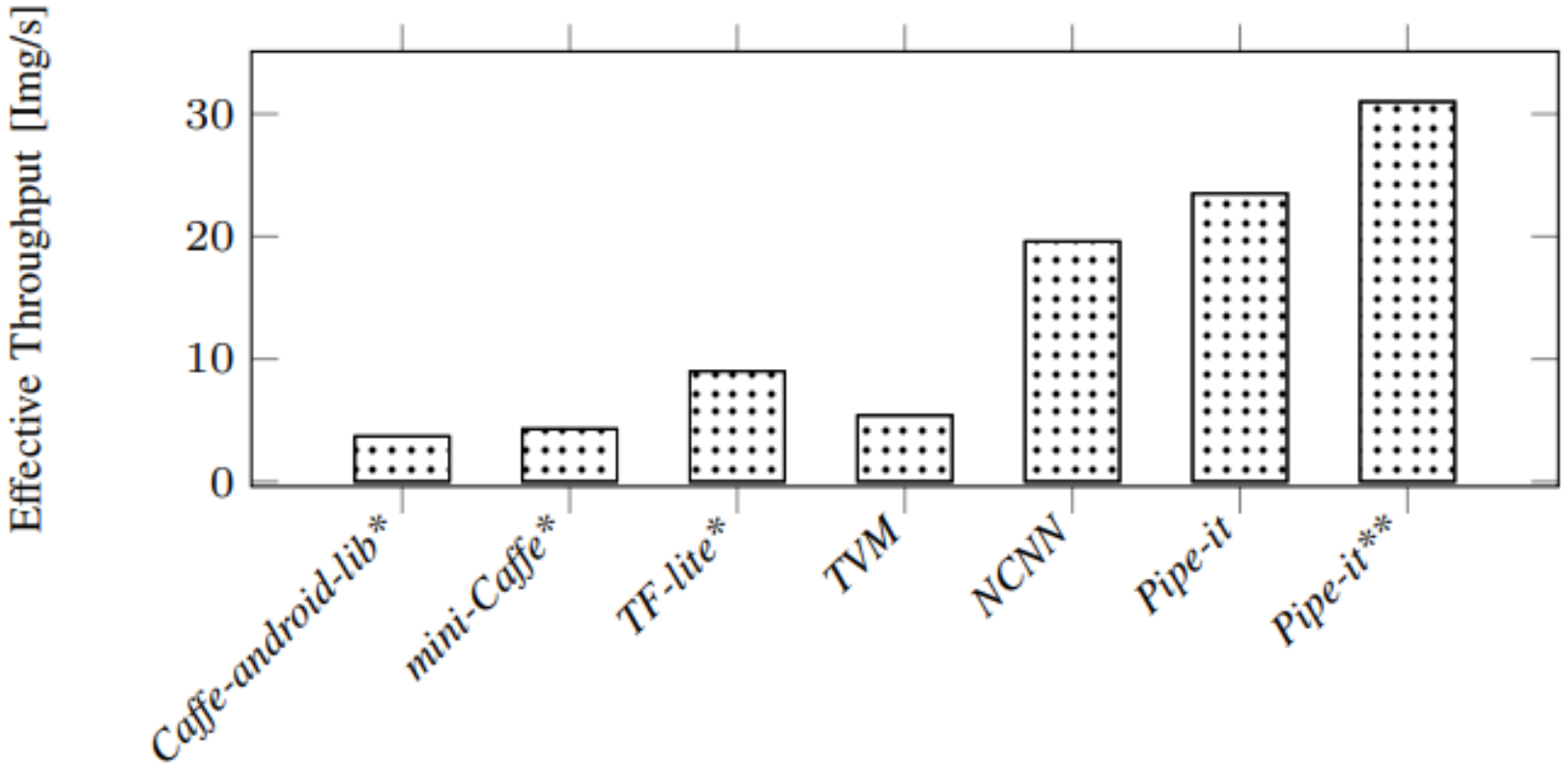






Một số công cụ khác

- Pocket flow: https://github.com/Tencent/PocketFlow
- Tencent NCNN: https://github.com/Tencent/ncnn



Tài liệu tham khảo

1. Bài giảng biên soạn dựa trên khóa cs231n của Stanford, bài giảng "**Deep Learning Hardware and Software**":

http://cs231n.stanford.edu

2. Tensorflow vs Keras vs PyTorch: https://databricks.com/session/a-tale-of-three-deep-learning-frameworks-tensorflow-keras-pytorch

3. NVIDIA TensorRT:

Fast Neural Network Inference with TensorRT on Autonomous

4. ARM chip:

Design And Reuse 2018 Keynote