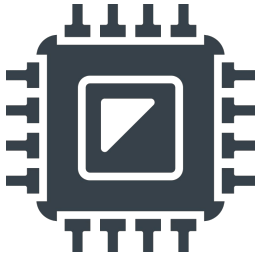
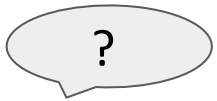
Deep Learning Compilers with PyTorch and LLVM MLIR

David Gens
http://david.g3ns.de
Cerebras Systems Inc.

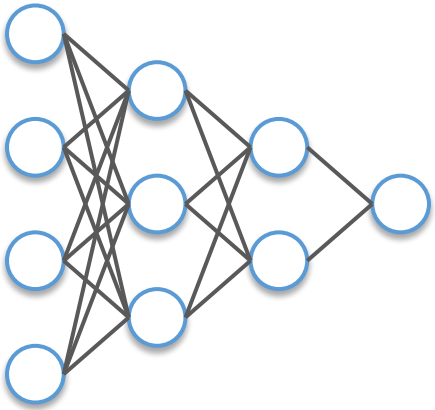
March, 2023

Deep Learning





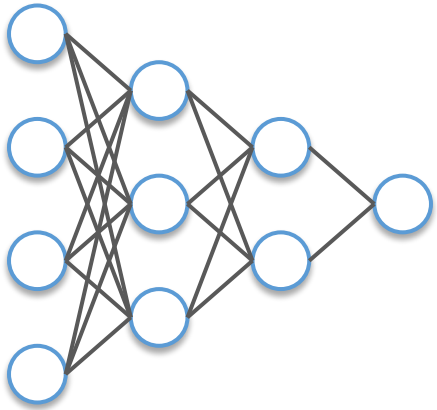
Deep Learning



Deep Learning

x

F



y

Dog

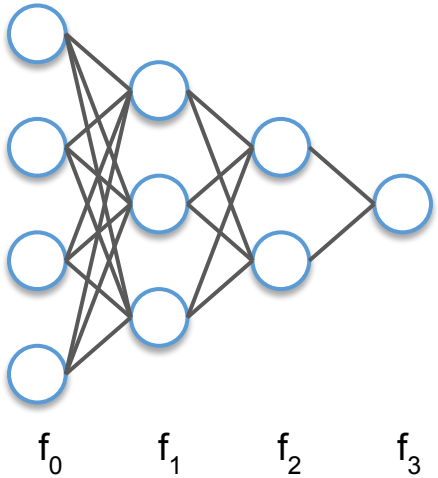
$$F(x) = f_3(f_2(f_1(f_0(x)) = y$$

Deep Learning

x

```
113 127 164 228
189 237 101 115
076 111 225 192
176 221 242 155
246 164 132 214
```

F



y

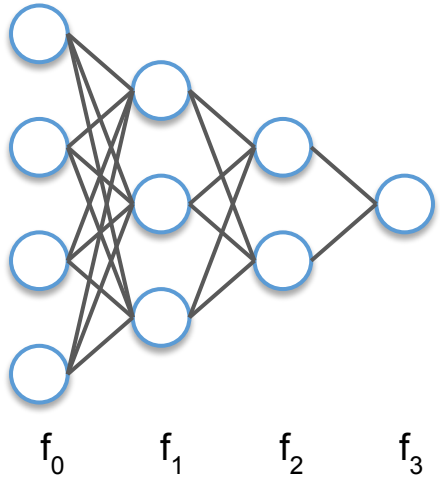
$$F(x) = f_3(f_2(f_1(f_0(x)) = y$$

Deep Learning

x

```
113 127 164 228
189 237 101 115
076 111 225 192
176 221 242 155
246 164 132 214
```

F



y

0
0
0
1
0
0
0
0
0
0

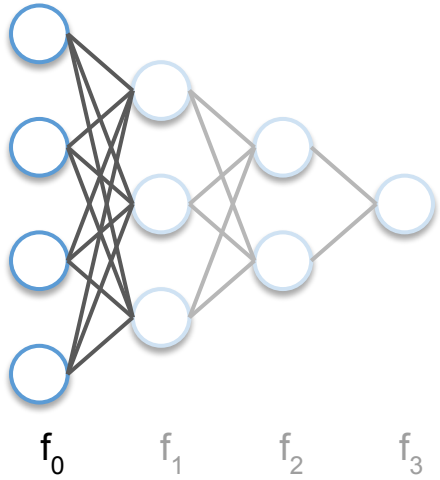
$$F(x) = f_3(f_2(f_1(f_0(x)) = y$$

Deep Learning

x

```
113 127 164 228
189 237 101 115
076 111 225 192
176 221 242 155
246 164 132 214
```

F



y

0
0
0
1
0
0
0
0
0
0

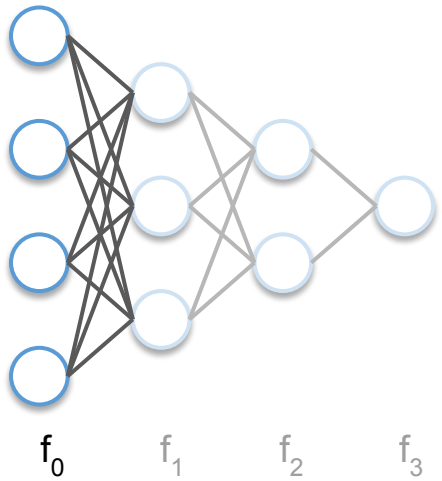
$$F(x) = f_3(f_2(f_1(f_0(x)) = y$$

Deep Learning

x

```
113 127 164 228
189 237 101 115
076 111 225 192
176 221 242 155
246 164 132 214
```

F



y

0
0
0
1
0
0
0
0
0
0

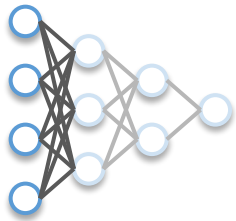
$$f_0(x) = a(W_0 \cdot x + b_0) \quad // \ In_{f0} = 4, \ Out_{f0} = 3$$

Deep Learning

W_0 x b_0

$$\begin{pmatrix} 0.27 & 1.00 & -0.39 & -0.79 \\ 0.60 & 0.19 & -0.26 & -0.48 \\ 0.12 & -0.39 & 0.54 & -0.20 \end{pmatrix} \begin{pmatrix} 0.68 \\ 0.50 \\ -0.05 \\ 0.11 \end{pmatrix} + \begin{pmatrix} 0.31 \\ -0.91 \\ 0.63 \end{pmatrix}$$

f_0



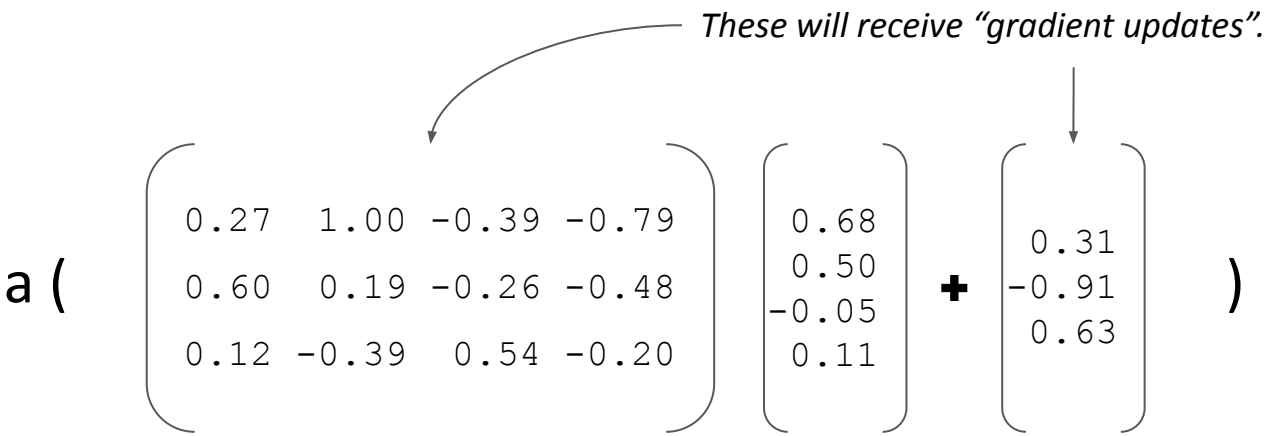
$$f_0(x) = a(W_0 \cdot x + b_0) \quad // \; In_{f0} = 4, \; Out_{f0} = 3$$

Deep Learning

$$a\left(\begin{bmatrix} 0.27 & 1.00 & -0.39 & -0.79 \\ 0.60 & 0.19 & -0.26 & -0.48 \\ 0.12 & -0.39 & 0.54 & -0.20 \end{bmatrix} \begin{bmatrix} 0.68 \\ 0.50 \\ -0.05 \\ 0.11 \end{bmatrix} + \begin{bmatrix} 0.31 \\ -0.91 \\ 0.63 \end{bmatrix}\right)$$

$$f_0(x) = a(W_0 \cdot x + b_0) \quad // \; In_{f0} = 4, \; Out_{f0} = 3$$

Deep Learning



$$f_0(x) = a(W_0 \cdot x + b_0) \quad // \; In_{f0} = 4, \; Out_{f0} = 3$$

Deep Learning

a (

```
f = [0 for _ in range(OUT)]
for o in range(OUT):
    for i in range(IN):
        f[o] += W[o][i]*x[i]
for o in range(OUT):
    f[o] += b[o]
```

)

$f_0(x) = a(W_0 \cdot x + b_0)$ // $In_{f0} = 4$, $Out_{f0} = 3$

Deep Learning

```
f = [0 for _ in range(OUT)]
for o in range(OUT):
    for i in range(IN):
        f[o] += W[o][i]*x[i]
for o in range(OUT):
    f[o] += b[o]
for o in range(OUT):
    f[o] = max(0,f[o])
```

$$f_0(x) = a(W_0 \cdot x + b_0) \quad // \; In_{f0} = 4, \; Out_{f0} = 3$$

Deep Learning

```
for f_i, f_o, W, b in layers:
    for o in range(len(f_o)):
        for i in range(len(f_i)):
            f_o[o] += W[o][i]*f_i[i]
    for o in range(len(f_o)):
        f_o[o] += b[o]
    for o in range(len(f_o)):
        f_o[o] = max(0,f_o[o])
```

Deep Learning

```
for x in samples:
    for f_i, f_o, W, b in layers:
        for o in range(len(f_o)):
            for i in range(len(f_i)):
                f_o[o] += W[o][i]*f_i[i]
        for o in range(len(f_o)):
            f_o[o] += b[o]
        for o in range(len(f_o)):
            f_o[o] = max(0,f_o[o])
```

Deep Learning

```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

Deep Learning

```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

These will receive "gradient updates".

Deep Learning

```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

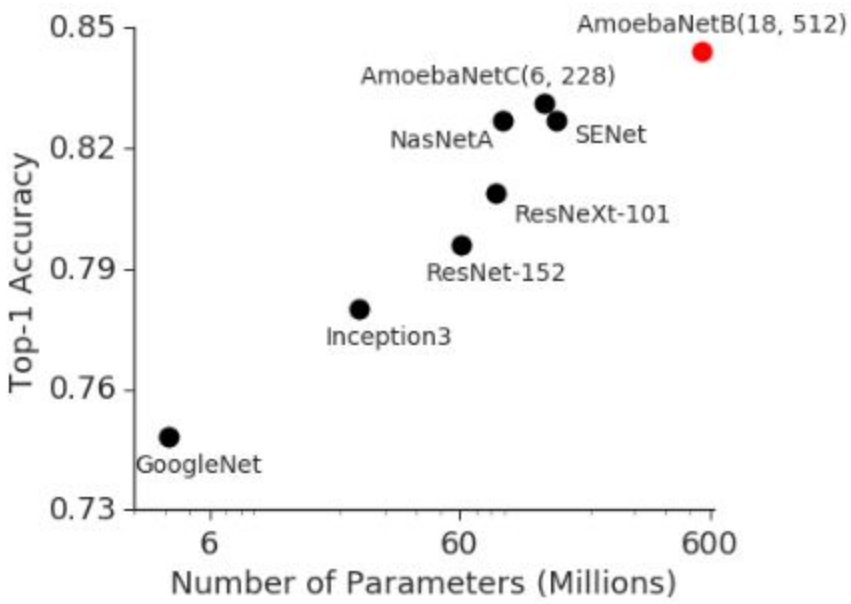
e.g., 100 epochs, 1M samples, 100 layers, dims 2 to 2^{16}

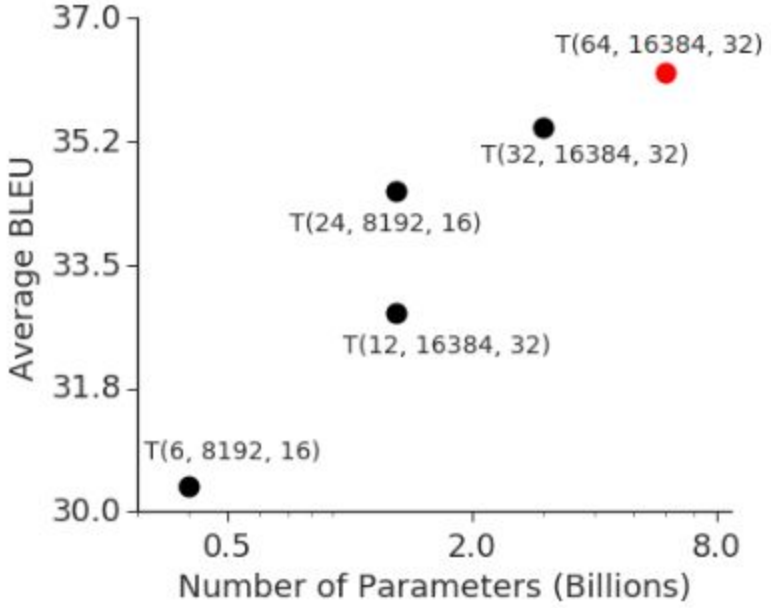
Deep Learning

```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

~ 2023 SOTA Workloads: ***10^{18} to 10^{21} FLOPs***

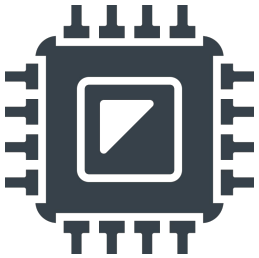
Deep Learning

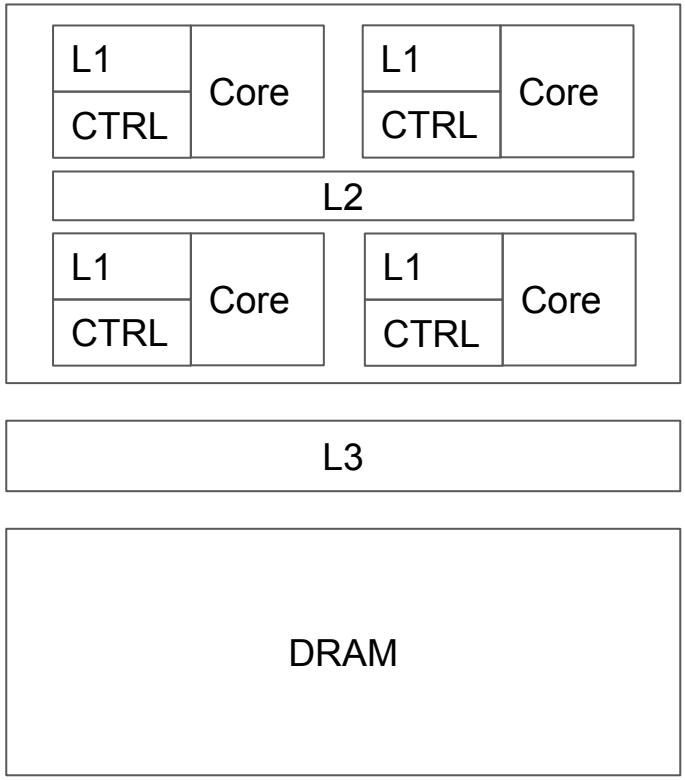




[1] GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism - Huang, et al., NeurIPS, 2019.

Target Architecture: CPU

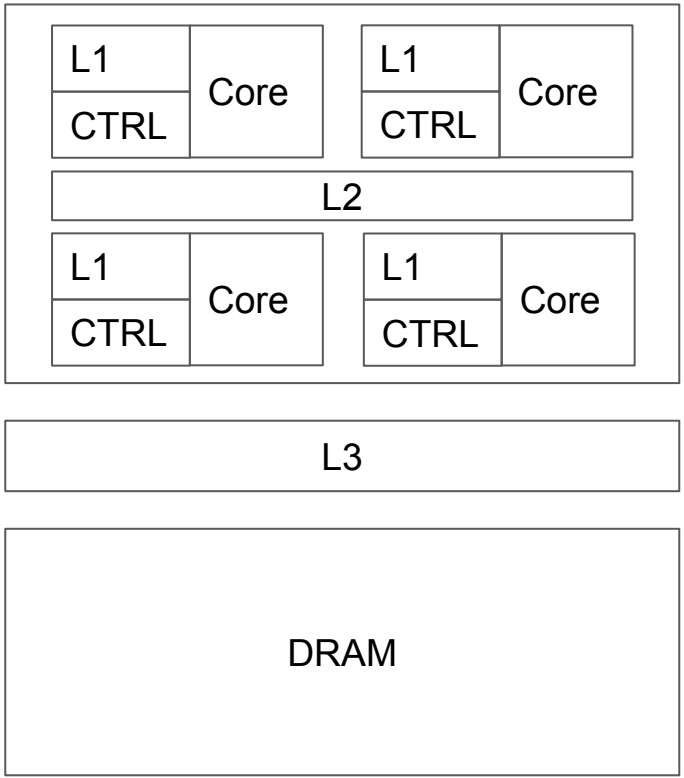




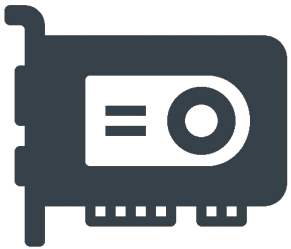
Target Architecture: CPU

- 4-32 “big” cores (8-64 threads)
- 128 GB Memory
- 2 TFLOPS peak
- Powerful OoO Engine
  - Branch Predictors
  - Prefetching
  - Reordering
  - Speculation
- Complex caches
- 100 GB/s memory bandwidth

Key Metric: **Integer** Performance!

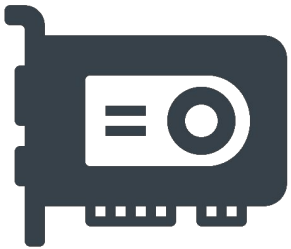


Target Architecture: GPU



```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

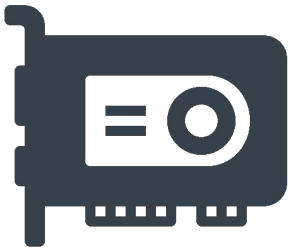
Target Architecture: GPU

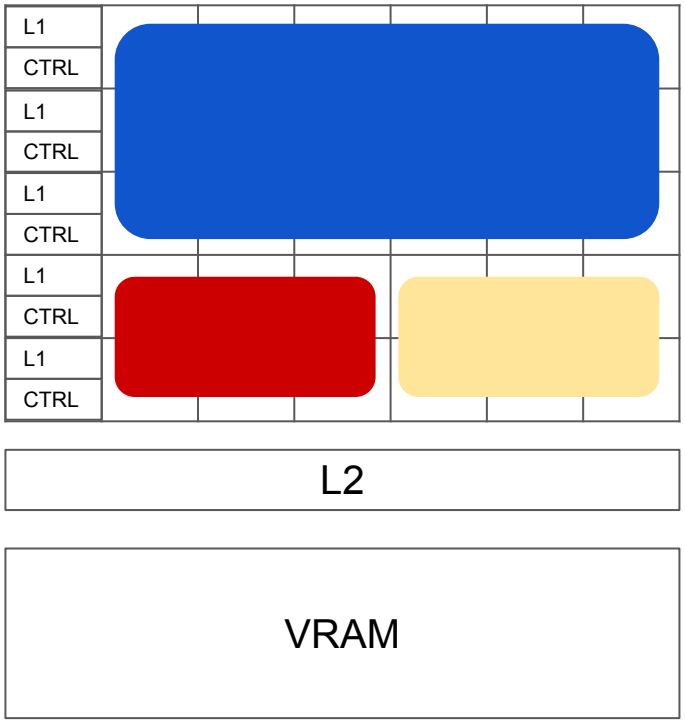


```
for _ in epochs:
    for x in samples:
        for f_i, f_o, W, b in layers:
            for o in range(len(f_o)):
                for i in range(len(f_i)):
                    f_o[o] += W[o][i]*f_i[i]
            for o in range(len(f_o)):
                f_o[o] += b[o]
            for o in range(len(f_o)):
                f_o[o] = max(0,f_o[o])
```

Kernels

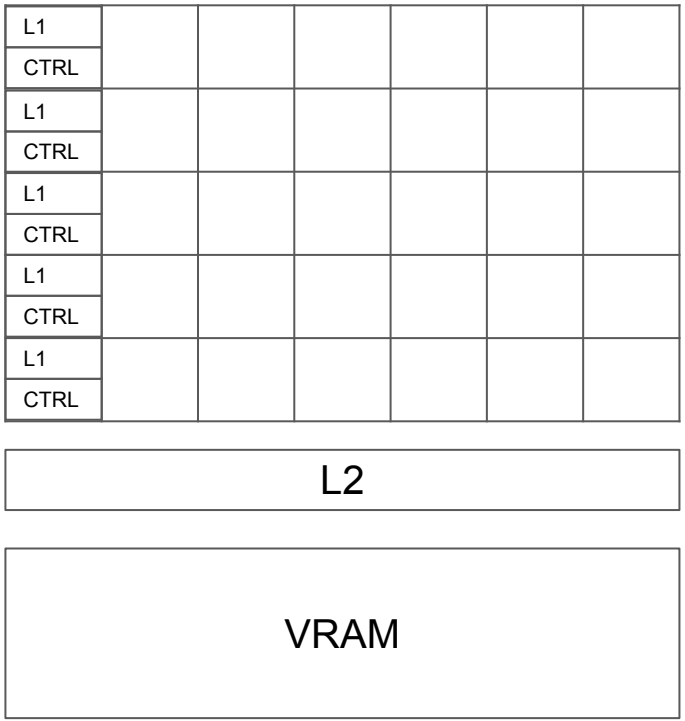
Target Architecture: GPU





Target Architecture: GPU

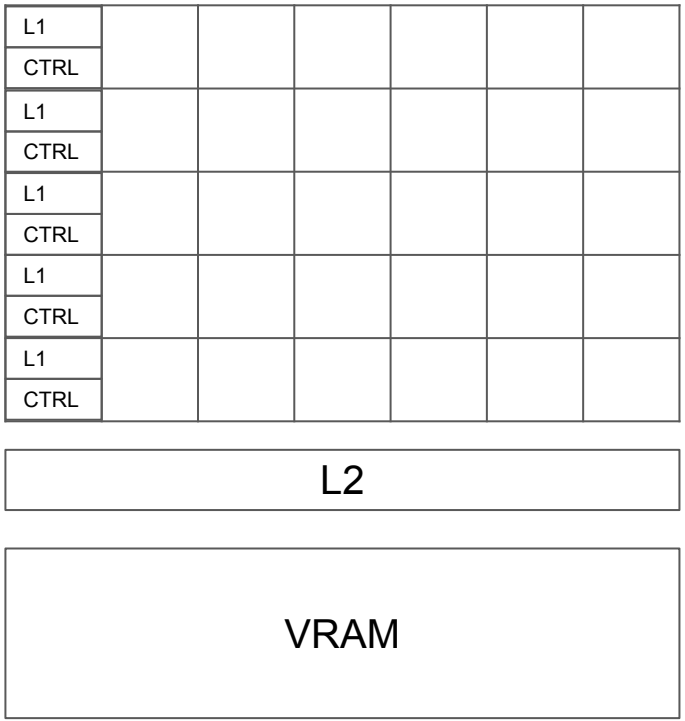
- 1000 cores
- 16 GB Memory
- 125 TFLOPS peak
- Groups of small cores execute the same operation on different data in lockstep
- 900 GB/s memory bandwidth



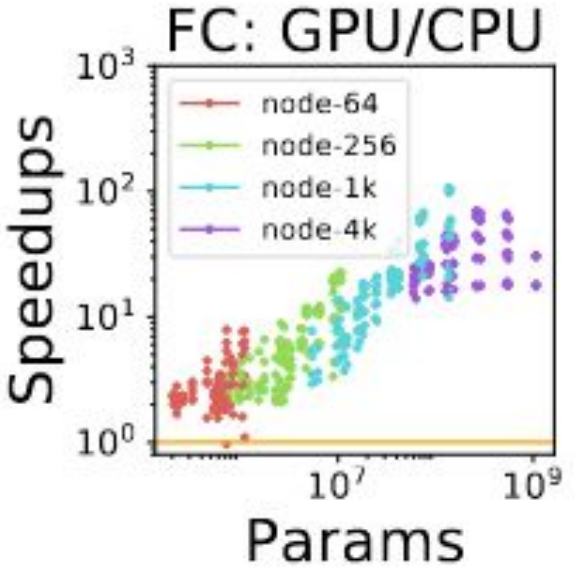
Target Architecture: GPU

- 1000 cores
- 16 GB Memory
- 125 TFLOPS peak
- Groups of small cores execute the same operation on different data in lockstep
- 900 GB/s memory bandwidth

Key Metric: **Floating Point** Performance!

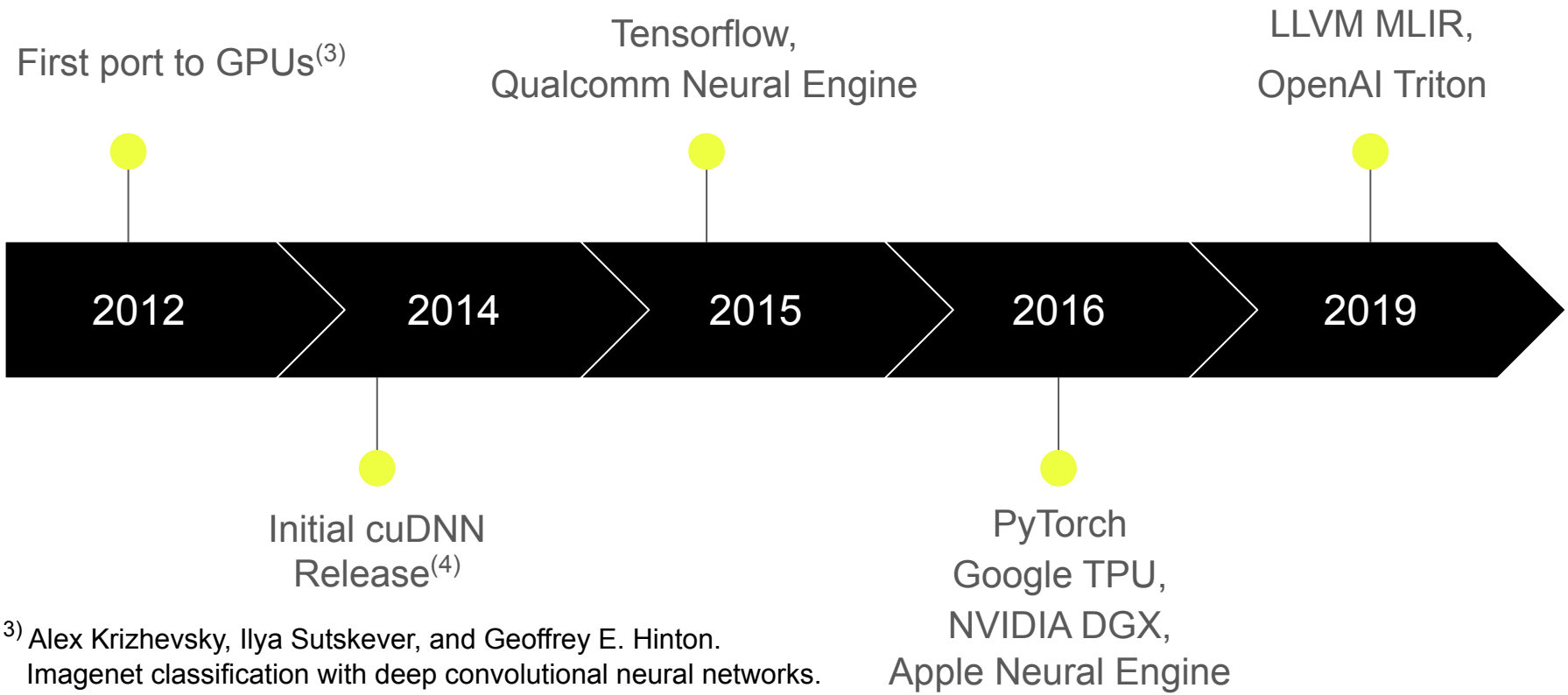


Speedup Comparison



(b) Node

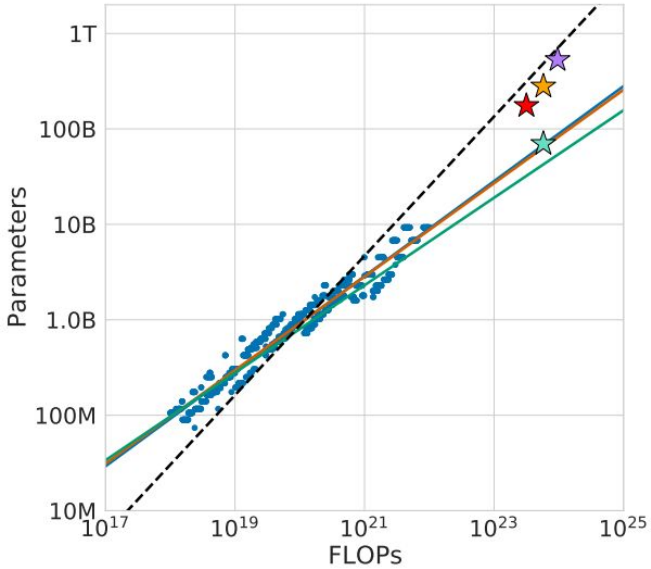
[2)] Wang, Yu Emma, Gu-Yeon Wei, and David Brooks. Figure 9 in "Benchmarking TPU, GPU, and CPU platforms for deep learning." Preprint on arXiv, 2019.



3) Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1106–1114, 2012.

4) cuDNN: Efficient Primitives for Deep Learning - Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, Evan Shelhamer. Preprint on Arxiv, 2014.

The Cost of Training

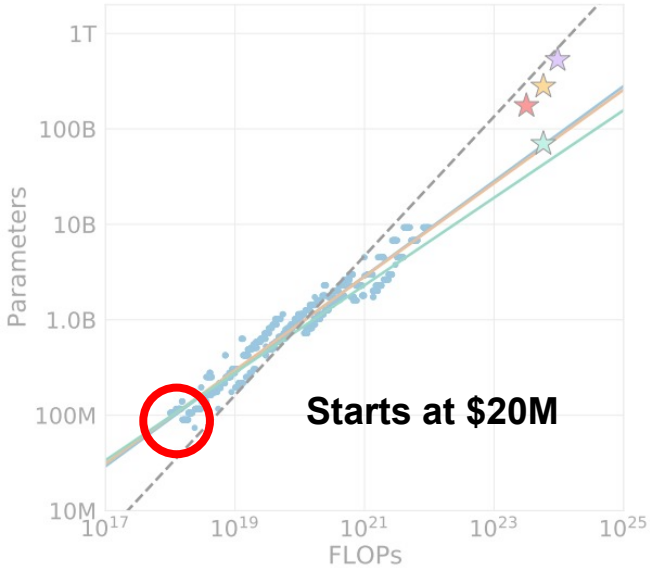


$$L(N, D) = \underbrace{\frac{A}{N^{\alpha}}}_{\text{finite model}} + \underbrace{\frac{B}{D^{\beta}}}_{\text{finite data}} + \underbrace{E}_{\text{irreducible}}$$

[5] Scaling Laws for Neural Language Models - Kaplan, et al., Preprint on Arxiv, 2020.

[6] Training Compute-Optimal Large Language Models - Hoffmann, Borgeaud, Mensch, et al., Preprint on Arxiv, 2022.

The Cost of Training



$$L(N, D) = \underbrace{\frac{A}{N^{\alpha}}}_{\text{finite model}} + \underbrace{\frac{B}{D^{\beta}}}_{\text{finite data}} + \underbrace{E}_{\text{irreducible}}$$

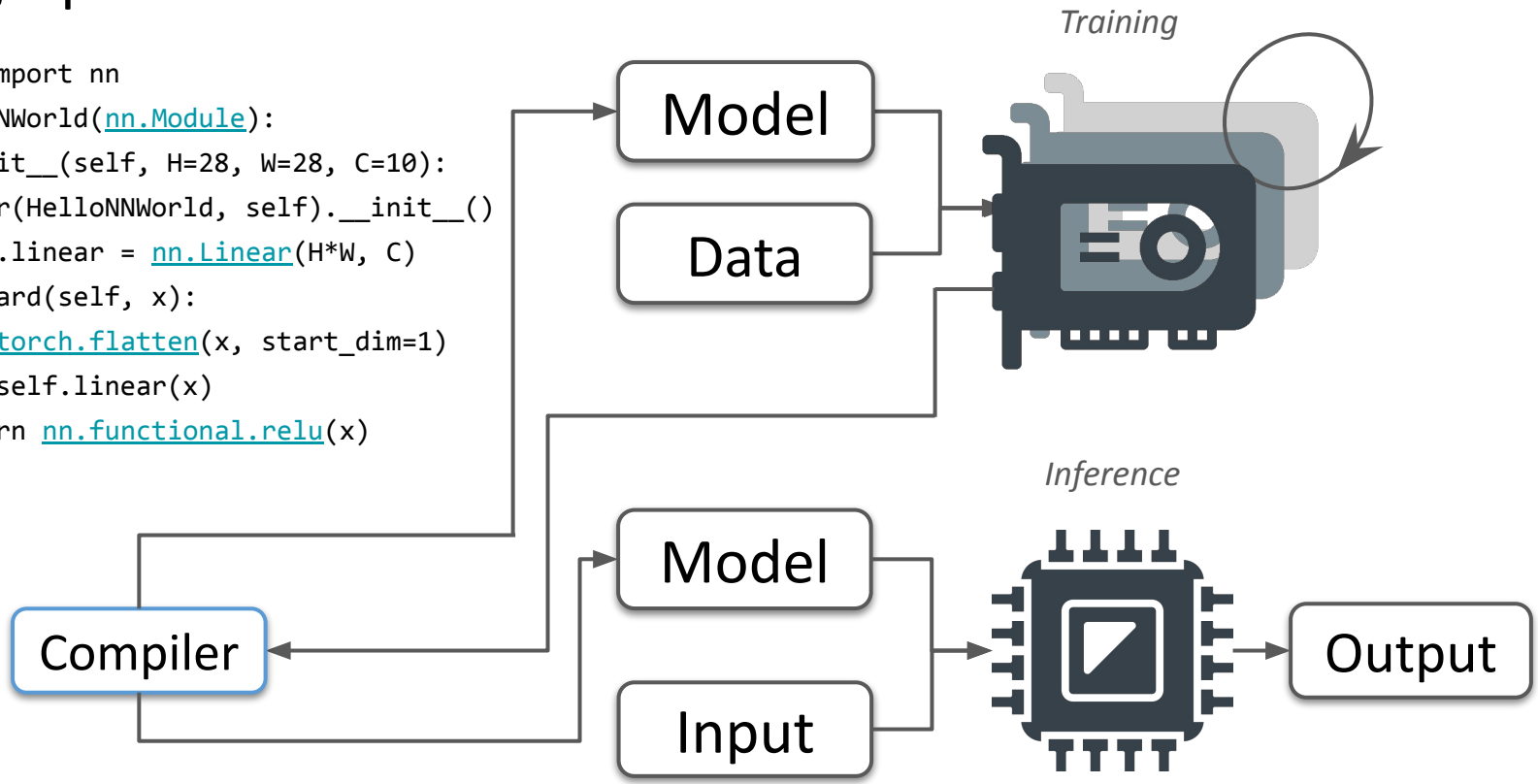
5) Scaling Laws for Neural Language Models - Kaplan, et al., Preprint on Arxiv, 2020.

6) Training Compute-Optimal Large Language Models - Hoffmann, Borgeaud, Mensch, et al., Preprint on Arxiv, 2022.

7) https://en.wikipedia.org/wiki/FLOPS#Hardware_costs (retrieved January 2023).

Scaling Up!

```
from torch import nn
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)
```



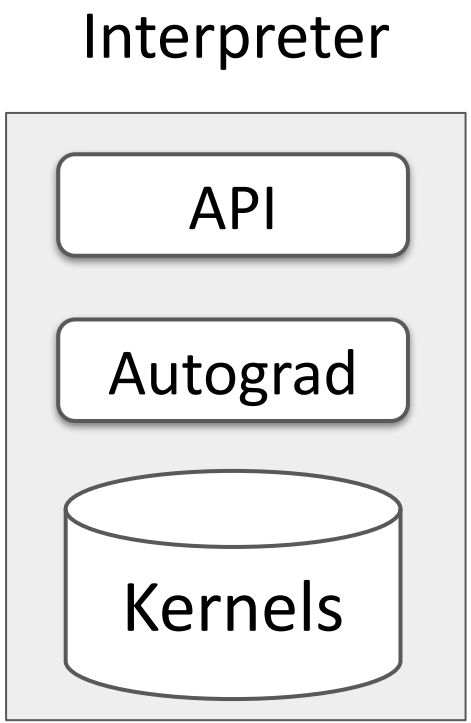
A Golden Age of Compilers?

“The Golden Age of Compiler Design in an Era of HW/SW Co-design”
ASPLOS Keynote 2021 by Chris Lattner
https://www.youtube.com/watch?v=4HgShra-KnY

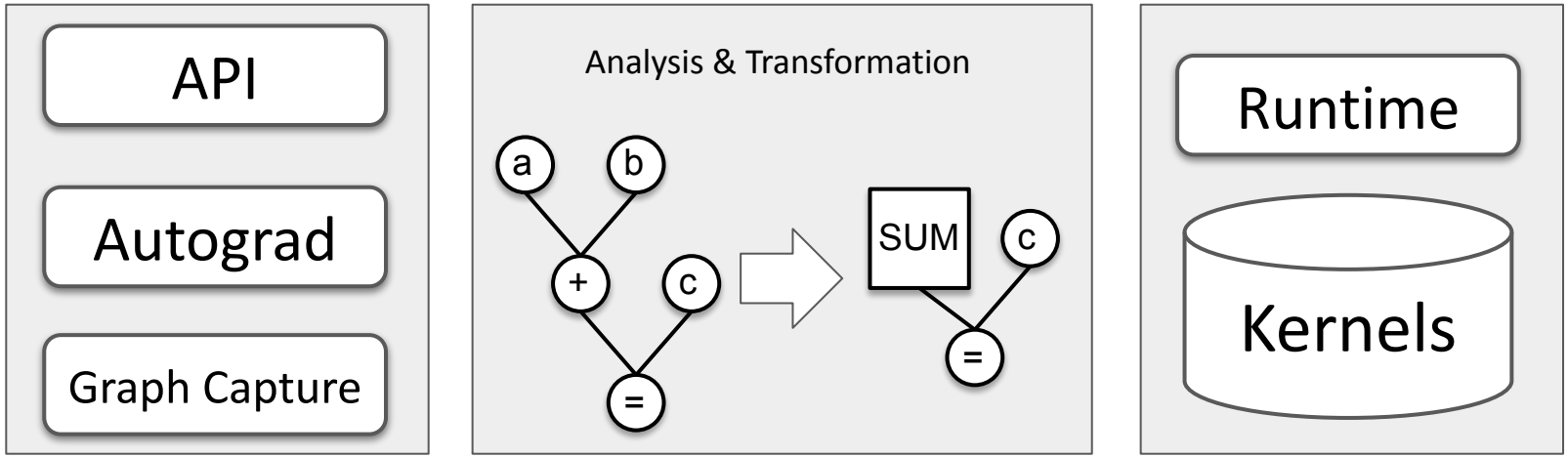
Deep Learning Frameworks

(mostly) Pytorch & Tensorflow

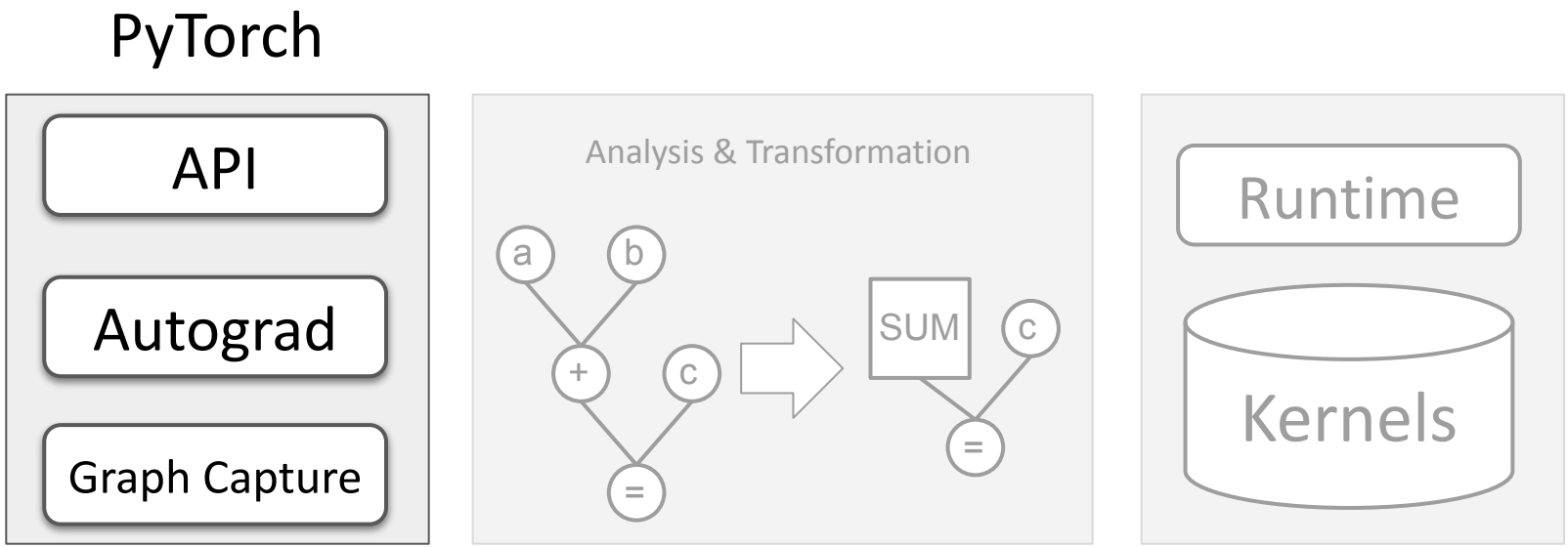
- Interpreters with pre-defined native kernels (“Eager Execution”)
- Target Focus: GPUs, TPUs
- Ongoing efforts to develop fast Just-in-Time engines
  - has proven challenging



Deep Learning Compilers



Deep Learning Compilers



Graph Capture

```
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)

def train(input, model, loss_fn, optimizer):
    model.train()
    for (X, y) in input:
        pred = model(X)
        loss = loss_fn(pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Graph Capture

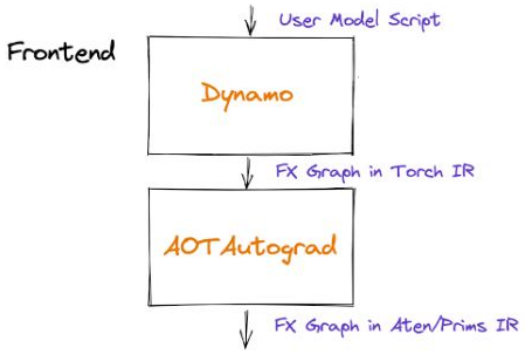
```
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)

def train(input, model, loss_fn, optimizer):
    model.train()
    for (X, y) in input:
        pred = model(X)
        loss = loss_fn(pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Graph Capture

```
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)

def train(input, model, loss_fn, optimizer):
    model.train()
    for (X, y) in input:
        pred = model(X)
        loss = loss_fn(pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Graph Capture

```
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)
```



https://pytorch.org/get-started/pytorch-2.0

```
forward():
    empty  = torch.ops.aten.empty.memory_format([10, 784], device = device(type='cpu'), pin_memory = False)
    detach = torch.ops.aten.detach.default(empty)
    detach_1 = torch.ops.aten.detach.default(detach)
    empty_1  = torch.ops.aten.empty.memory_format([10], device = device(type='cpu'), pin_memory = False)
    detach_2 = torch.ops.aten.detach.default(empty_1)
    detach_3 = torch.ops.aten.detach.default(detach_2)
    uniform_ = torch.ops.aten.uniform_.default(detach_1, -0.03571428571428571, 0.03571428571428571)
    uniform__1 = torch.ops.aten.uniform_.default(detach_3, -0.03571428571428571, 0.03571428571428571)
    view = torch.ops.aten.view.default(arg0_1, [16, 784])
    t = torch.ops.aten.t.default(uniform_)
    addmm = torch.ops.aten.addmm.default(uniform__1, view, t)
    relu = torch.ops.aten.relu.default(addmm)
    return relu
```

Graph Capture

```
class HelloNNWorld(nn.Module):
    def __init__(self, H=28, W=28, C=10):
        super(HelloNNWorld, self).__init__()
        self.linear = nn.Linear(H*W, C)
    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.linear(x)
        return nn.functional.relu(x)

def train(input, model, loss_fn, optimizer):
    model.train()
    for (X, y) in input:
        pred = model(X)
        loss = loss_fn(pred, y)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Graph Capture

```
forward+loss+backward():
    empty  = torch.ops.aten.empty.memory_format([10, 784], device = device(type='cpu'), pin_memory = False)
    detach = torch.ops.aten.detach.default(empty)
    detach_1 = torch.ops.aten.detach.default(detach)
    empty_1  = torch.ops.aten.empty.memory_format([10], device = device(type='cpu'), pin_memory = False)
    detach_2 = torch.ops.aten.detach.default(empty_1)
    detach_3 = torch.ops.aten.detach.default(detach_2)
    uniform_ = torch.ops.aten.uniform_.default(detach_1, -0.03571428571428571, 0.03571428571428571)
    uniform__1 = torch.ops.aten.uniform_.default(detach_3, -0.03571428571428571, 0.03571428571428571)
    view = torch.ops.aten.view.default(arg0_1, [16, 784])
    t = torch.ops.aten.t.default(uniform_)
    addmm = torch.ops.aten.addmm.default(uniform__1, view, t)
    relu = torch.ops.aten.relu.default(addmm)
    detach_4 = torch.ops.aten.detach.default(relu)
    _log_softmax = torch.ops.aten._log_softmax.default(relu, 1, False)
    detach_5 = torch.ops.aten.detach.default(_log_softmax)
    nll_loss_forward = torch.ops.aten.nll_loss_forward.default(_log_softmax, arg1_1, None, 1, -100)
    getitem = nll_loss_forward[0]
    getitem_1 = nll_loss_forward[1]
```

Graph Capture

```
ones_like = torch.ops.aten.ones_like.default(getitem, dtype = torch.float32, layout = torch.strided,
              device = device(type='cpu'), pin_memory = False, memory_format = torch.preserve_format)
nll_loss_backward = torch.ops.aten.nll_loss_backward.default(ones_like, _log_softmax, arg1_1, None, 1,
                      -100, getitem_1)
detach_6 = torch.ops.aten.detach.default(detach_5)
_log_softmax_backward_data = torch.ops.aten._log_softmax_backward_data.default(nll_loss_backward,
                               detach_6, 1, torch.float32)
detach_7 = torch.ops.aten.detach.default(detach_4)
threshold_backward = torch.ops.aten.threshold_backward.default(_log_softmax_backward_data, detach_7, 0)
t_1 = torch.ops.aten.t.default(threshold_backward)
mm = torch.ops.aten.mm.default(t_1, view)
t_2 = torch.ops.aten.t.default(mm)
sum_1 = torch.ops.aten.sum.dim_IntList(threshold_backward, [0], True)
view_1 = torch.ops.aten.view.default(sum_1, [10])
detach_8 = torch.ops.aten.detach.default(view_1)
detach_9 = torch.ops.aten.detach.default(detach_8)
t_3 = torch.ops.aten.t.default(t_2)
detach_10 = torch.ops.aten.detach.default(t_3)
detach_11 = torch.ops.aten.detach.default(detach_10)
return [getitem]
```

PyTorch - Tensor API, Dynamic Dispatch, Autograd

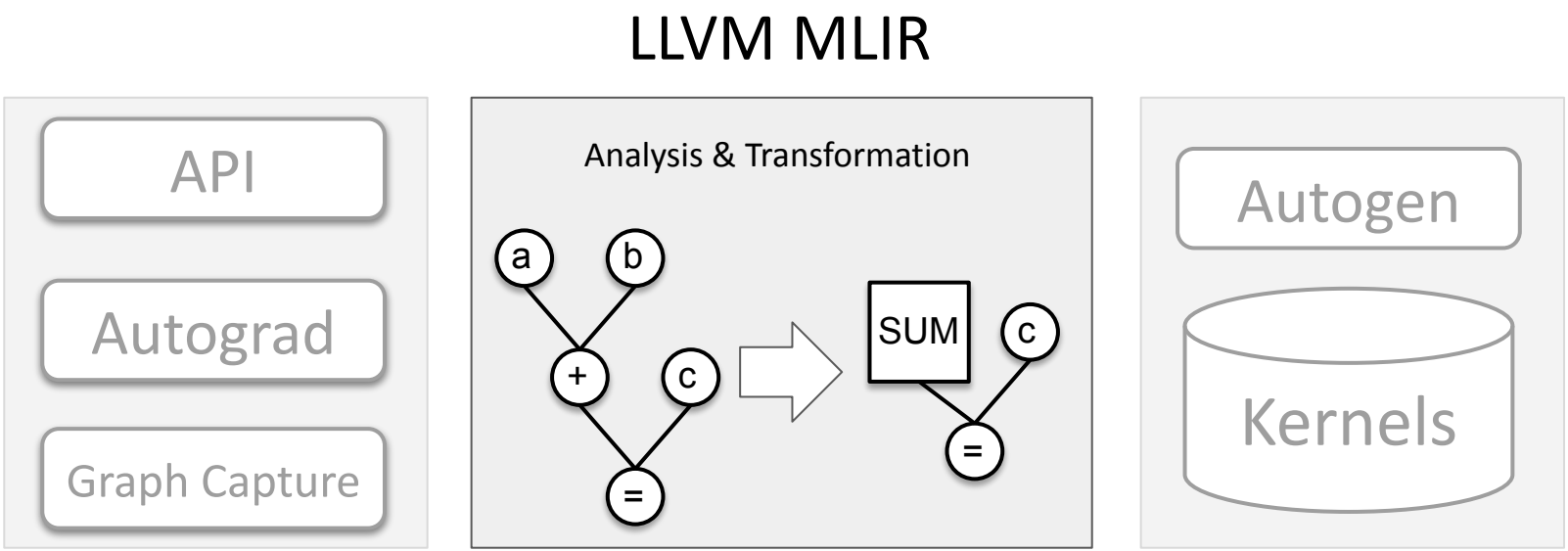
Cool! Wait, what about the Weights? In what format does it expect to receive the Activations? How do I know what Kernels are actually being called? What types can I pass in?

https://github.com/pytorch/pytorch/blob/master/aten/src/ATen/native/native_functions.yaml#L3979

```
- func: native_batch_norm(
          Tensor input, Tensor? weight, Tensor? Bias,
          Tensor? running_mean, Tensor? Running_var,
          bool training, float momentum, float eps
  ) -> (Tensor, Tensor, Tensor)
  dispatch:
    CPU: batch_norm_cpu
    CUDA: batch_norm_cuda
    MPS: batch_norm_mps
    MkldnnCPU: mkldnn_batch_norm
  tags: core
```

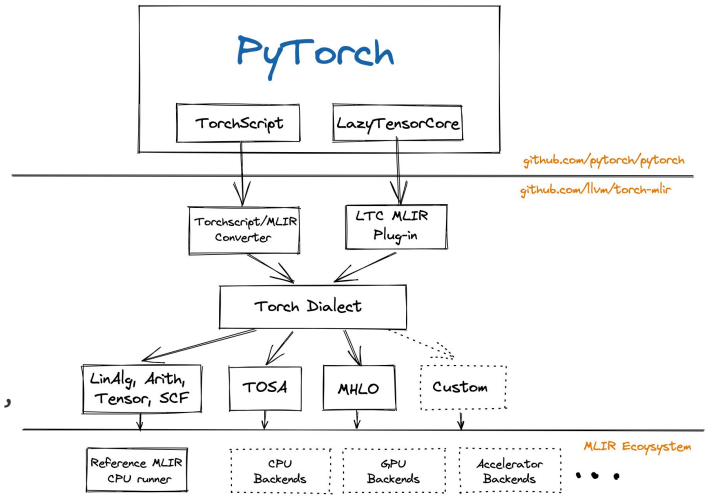
Recommended Reading: http://blog.ezyang.com/2019/05/pytorch-internals/

Deep Learning Compilers



LLVM MLIR!

https://github.com/llvm/torch-mlir




```
module attributes {torch.debug_module_name = "GraphModule"} {
  func.func @forward(
     %arg0: !torch.vtensor<[10,784],f32>,
     %arg1: !torch.vtensor<[10],f32>,
     %arg2: !torch.vtensor<[16,28,28],f32>) -> (!torch.vtensor<[16,10],f32>,
           !torch.vtensor<[16,784],f32>,!torch.vtensor<[16,10],f32>)
  {
    %int0   = torch.constant.int 0
    %int1   = torch.constant.int 1
    %int784 = torch.constant.int 784
    %int16  = torch.constant.int 16
    %0 = torch.prim.ListConstruct %int16, %int784 : (!torch.int, !torch.int) -> !torch.list<int>
    %1 = torch.aten.view %arg2, %0 : !torch.vtensor<[16,28,28],f32>, !torch.list<int> -> !torch.vtensor<[16,784],f32>
    %2 = torch.aten.transpose.int %arg0, %int0, %int1 : !torch.vtensor<[10,784],f32>, !torch.int, !torch.int
                                                     -> !torch.vtensor<[784,10],f32>
    %3 = torch.aten.mm %1, %2 : !torch.vtensor<[16,784],f32>, !torch.vtensor<[784,10],f32> -> !torch.vtensor<[16,10],f32>
    %4 = torch.aten.mul.Scalar %arg1, %int1 : !torch.vtensor<[10],f32>, !torch.int -> !torch.vtensor<[10],f32>
    %5 = torch.aten.add.Tensor %4, %3, %int1 : !torch.vtensor<[10],f32>, !torch.vtensor<[16,10],f32>, !torch.int
                                          -> !torch.vtensor<[16,10],f32>
    %6 = torch.aten.relu %5 : !torch.vtensor<[16,10],f32> -> !torch.vtensor<[16,10],f32>
    return %6, %1, %6 : !torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>, !torch.vtensor<[16,10],f32>
  }
}
```

MLIR - Textual Format

```
module attributes {torch.debug_module_name = "GraphModule"} {
  func.func @forward(
     %arg0: !torch.vtensor<[10,784],f32>,
     %arg1: !torch.vtensor<[10],f32>,
     %arg2: !torch.vtensor<[16,28,28],f32>
  ) -> (!torch.vtensor<[16,10],f32>,!torch.vtensor<[16,784],f32>,!torch.vtensor<[16,10],f32>)
  {
    %int0   = torch.constant.int 0
    %int1   = torch.constant.int 1
    %int784 = torch.constant.int 784
    %int16  = torch.constant.int 16
    %0 = torch.prim.ListConstruct %int16, %int784 : (!torch.int, !torch.int) -> !torch.list<int>
    %1 = torch.aten.view %arg2, %0 : !torch.vtensor<[16,28,28],f32>, !torch.list<int> -> !torch.vtensor<[16,784],f32>
    %2 = torch.aten.transpose.int %arg0, %int0, %int1 : !torch.vtensor<[10,784],f32>, !torch.int, !torch.int
                                                      -> !torch.vtensor<[784,10],f32>
    %3 = torch.aten.mm %1, %2 : !torch.vtensor<[16,784],f32>, !torch.vtensor<[784,10],f32> -> !torch.vtensor<[16,10],f32>
    %4 = torch.aten.mul.Scalar %arg1, %int1 : !torch.vtensor<[10],f32>, !torch.int -> !torch.vtensor<[10],f32>
    %5 = torch.aten.add.Tensor %4, %3, %int1 : !torch.vtensor<[10],f32>, !torch.vtensor<[16,10],f32>, !torch.int
                                            -> !torch.vtensor<[16,10],f32>
    %6 = torch.aten.relu %5 : !torch.vtensor<[16,10],f32> -> !torch.vtensor<[16,10],f32>
    return %6, %1, %6 : !torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>, !torch.vtensor<[16,10],f32>
  }
}
```

MLIR - Modules, Functions

```
module attributes {torch.debug_module_name = "GraphModule"} {
  func.func @forward(
     %arg0: !torch.vtensor<[10,784],f32>,
     %arg1: !torch.vtensor<[10],f32>,
     %arg2: !torch.vtensor<[16,28,28],f32>
  ) -> (!torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>,!torch.vtensor<[16,10],f32>)
  {
    %int0   = torch.constant.int 0
    %int1   = torch.constant.int 1
    %int784 = torch.constant.int 784
    %int16  = torch.constant.int 16
    %0 = torch.prim.ListConstruct %int16, %int784 : (!torch.int, !torch.int) -> !torch.list<int>
    %1 = torch.aten.view %arg2, %0 : !torch.vtensor<[16,28,28],f32>, !torch.list<int> -> !torch.vtensor<[16,784],f32>
    %2 = torch.aten.transpose.int %arg0, %int0, %int1 : !torch.vtensor<[10,784],f32>, !torch.int, !torch.int
                                                      -> !torch.vtensor<[784,10],f32>
    %3 = torch.aten.mm %1, %2 : !torch.vtensor<[16,784],f32>, !torch.vtensor<[784,10],f32> -> !torch.vtensor<[16,10],f32>
    %4 = torch.aten.mul.Scalar %arg1, %int1 : !torch.vtensor<[10],f32>, !torch.int -> !torch.vtensor<[10],f32>
    %5 = torch.aten.add.Tensor %4, %3, %int1 : !torch.vtensor<[10],f32>, !torch.vtensor<[16,10],f32>, !torch.int
                                            -> !torch.vtensor<[16,10],f32>
    %6 = torch.aten.relu %5 : !torch.vtensor<[16,10],f32> -> !torch.vtensor<[16,10],f32>
    return %6, %1, %6 : !torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>, !torch.vtensor<[16,10],f32>
  }
}
```

MLIR - Types

```
module attributes {torch.debug_module_name = "GraphModule"} {
  func.func @forward(
     %arg0: !torch.vtensor<[10,784],f32>,
     %arg1: !torch.vtensor<[10],f32>,
     %arg2: !torch.vtensor<[16,28,28],f32>
  ) -> (!torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>,!torch.vtensor<[16,10],f32>)
  {
    %int0   = torch.constant.int 0
    %int1   = torch.constant.int 1
    %int784 = torch.constant.int 784
    %int16  = torch.constant.int 16
    %0 = torch.prim.ListConstruct %int16, %int784 : (!torch.int, !torch.int) -> !torch.list<int>
    %1 = torch.aten.view %arg2, %0 : !torch.vtensor<[16,28,28],f32>, !torch.list<int> -> !torch.vtensor<[16,784],f32>
    %2 = torch.aten.transpose.int %arg0, %int0, %int1 : !torch.vtensor<[10,784],f32>, !torch.int, !torch.int
                                                      -> !torch.vtensor<[784,10],f32>
    %3 = torch.aten.mm %1, %2 : !torch.vtensor<[16,784],f32>, !torch.vtensor<[784,10],f32> -> !torch.vtensor<[16,10],f32>
    %4 = torch.aten.mul.Scalar %arg1, %int1 : !torch.vtensor<[10],f32>, !torch.int -> !torch.vtensor<[10],f32>
    %5 = torch.aten.add.Tensor %4, %3, %int1 : !torch.vtensor<[10],f32>, !torch.vtensor<[16,10],f32>, !torch.int
                                            -> !torch.vtensor<[16,10],f32>
    %6 = torch.aten.relu %5 : !torch.vtensor<[16,10],f32> -> !torch.vtensor<[16,10],f32>
    return %6, %1, %6 : !torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>, !torch.vtensor<[16,10],f32>
  }
}
```

MLIR - Operations

```
module attributes {torch.debug_module_name = "GraphModule"} {
  func.func @forward(
     %arg0: !torch.vtensor<[10,784],f32>,
     %arg1: !torch.vtensor<[10],f32>,
     %arg2: !torch.vtensor<[16,28,28],f32>
  ) -> (!torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>,!torch.vtensor<[16,10],f32>)
  {
    %int0   = torch.constant.int 0
    %int1   = torch.constant.int 1
    %int784 = torch.constant.int 784
    %int16  = torch.constant.int 16
    %0 = torch.prim.ListConstruct %int16, %int784 : (!torch.int, !torch.int) -> !torch.list<int>
    %1 = torch.aten.view %arg2, %0 : !torch.vtensor<[16,28,28],f32>, !torch.list<int> -> !torch.vtensor<[16,784],f32>
    %2 = torch.aten.transpose.int %arg0, %int0, %int1 : !torch.vtensor<[10,784],f32>, !torch.int, !torch.int
                                                     -> !torch.vtensor<[784,10],f32>
    %3 = torch.aten.mm %1, %2 : !torch.vtensor<[16,784],f32>, !torch.vtensor<[784,10],f32> -> !torch.vtensor<[16,10],f32>
    %4 = torch.aten.mul.Scalar %arg1, %int1 : !torch.vtensor<[10],f32>, !torch.int -> !torch.vtensor<[10],f32>
    %5 = torch.aten.add.Tensor %4, %3, %int1 : !torch.vtensor<[10],f32>, !torch.vtensor<[16,10],f32>, !torch.int
                                          -> !torch.vtensor<[16,10],f32>
    %6 = torch.aten.relu %5 : !torch.vtensor<[16,10],f32> -> !torch.vtensor<[16,10],f32>
    return %6, %1, %6 : !torch.vtensor<[16,10],f32>, !torch.vtensor<[16,784],f32>, !torch.vtensor<[16,10],f32>
  }
}
```

Demo!

MLIR - Dialects

- Dialect Infrastructure
  - Human-readable Text format for SSA Graphs
  - Namespaces for Operations and Types
  - Basics are built-in
  - Boilerplate C++ classes and verifiers can be largely auto-generated from definitions using *TableGen*
- General Graph Transformations
  - Through “Pattern Rewrites”
- Translation between different dialects
  - Called “Conversion” - can be partial
  - E.g. “torch” -> “linalg” -> “LLVM” in theory enables lowering through LLVM IR to any target LLVM has a backend for
    - **In practice WIP!**

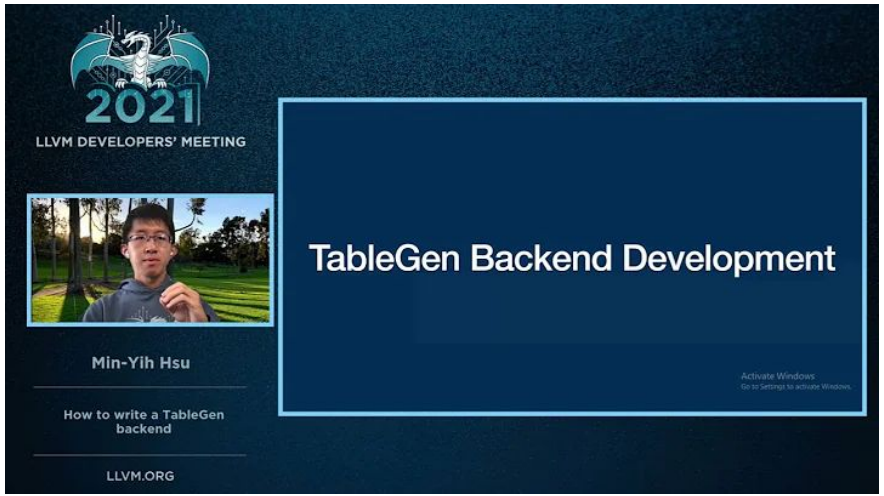
https://github.com/llvm/torch-mlir/blob/main/include/torch-mlir/Dialect/Torch/IR/GeneratedTorchOps.td#L4045

```
def Torch_AtenMatmulOp : Torch_Op<"aten.matmul", [
    AllowsTypeRefinement,
    HasValueSemantics,
    ReadOnly
  ]> {
  let summary = "Generated op for
                 `aten::matmul : (Tensor, Tensor) -> (Tensor)`";
  let arguments = (ins
    AnyTorchTensorType:$self,
    AnyTorchTensorType:$other
  );
  let results = (outs
    AnyTorchTensorType:$result
  );
  let hasCustomAssemblyFormat = 1;
  let extraClassDefinition = [{
    ParseResult AtenMatmulOp::parse(OpAsmParser &parser,
                                    OperationState &result) {
      return parseDefaultTorchOp(parser, result, 2, 1);
    }
    void AtenMatmulOp::print(OpAsmPrinter &printer) {
      printDefaultTorchOp(printer, *this, 2, 1);
    }
  }];
}
```

MLIR - Dialects

https://github.com/llvm/torch-mlir/blob/main/include/torch-mlir/Dialect/Torch/IR/GeneratedTorchOps.td#L4045

Watch Min's TableGen Talk!



https://www.youtube.com/watch?v=UP-LBRbvI_U

```
def Torch_AtenMatmulOp : Torch_Op<"aten.matmul", [
    AllowsTypeRefinement,
    HasValueSemantics,
    ReadOnly
  ]> {
  let summary = "Generated op for
                 `aten::matmul : (Tensor, Tensor) -> (Tensor)`";
  let arguments = (ins
    AnyTorchTensorType:$self,
    AnyTorchTensorType:$other
  );
  let results = (outs
    AnyTorchTensorType:$result
  );
  let hasCustomAssemblyFormat = 1;
  let extraClassDefinition = [{
    ParseResult AtenMatmulOp::parse(OpAsmParser &parser,
                                    OperationState &result) {
      return parseDefaultTorchOp(parser, result, 2, 1);
    }
    void AtenMatmulOp::print(OpAsmPrinter &printer) {
      printDefaultTorchOp(printer, *this, 2, 1);
    }
  }];
}
```

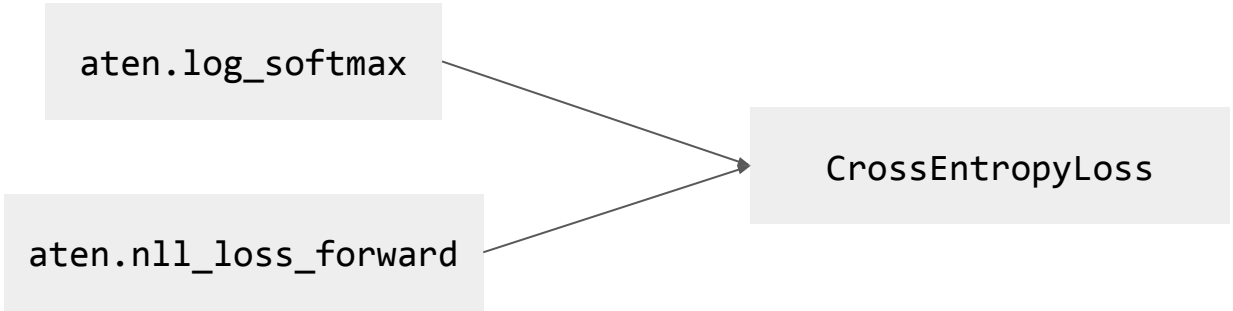
MLIR - Pattern Rewrites

- Generic DAG-to-DAG Transformations
  - Inspired by LLVM's instruction selection algorithm[8]
  - Can be specified declaratively or using C++
- Different algorithms for pattern application ("Drivers")
  - Cost models
  - Bottom-up, Top-Down, Greedy
  - Debugging, Filters, Diagnostics

8) https://eli.thegreenplace.net/2013/02/25/a-deeper-look-into-the-llvm-code-generator-part-1

MLIR - Pattern Rewrites

- e.g. `torch.nn.CrossEntropyLoss`[9]



9) https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html

MLIR - Pattern Rewrites

https://github.com/llvm/torch-mlir/blob/main/lib/Conversion/TorchToLinalg/Pooling.cpp#L128

```
class ConvertAtenMaxPool2dOp : public OpConversionPattern<AtenMaxPool2dOp> {
public:
  using OpConversionPattern::OpConversionPattern;
  LogicalResult
  matchAndRewrite(AtenMaxPool2dOp op, OpAdaptor adaptor,
                  ConversionPatternRewriter &rewriter) const override {
    if (failed(verifyLinalgCompatibleTypes(op, rewriter)))
      return failure();

    TypeConverter *typeConverter = getTypeConverter();
    Value self = adaptor.getSelf();
    int64_t selfRank = self.getType().cast<RankedTensorType>().getRank();
    // TODO: Add support for 3D inputs.
    if (selfRank == 3)
      return rewriter.notifyMatchFailure(
          op, "unimplemented: only support 4D input");

    bool ceilMode;
    SmallVector<Value, 2> kernelSizeIntValues;
    SmallVector<int64_t, 2> strideInts, paddingInts, dilationInts;
    if (!matchPattern(op.getDilation(), m_TorchListOfConstantInts(dilationInts)))
      return rewriter.notifyMatchFailure(op,
                                         "only support constant int dilations");
    if (failed(checkAndGetPoolingParameters<AtenMaxPool2dOp>(
            op, rewriter, typeConverter, ceilMode, kernelSizeIntValues,
            strideInts, paddingInts)))
      return rewriter.notifyMatchFailure(op, "invalid pooling parameters");

    Type elementType =
        self.getType().cast<RankedTensorType>().getElementType();
    auto smallestFPValueAttr = rewriter.getFloatAttr(
        elementType,
        APFloat::getLargest(
            elementType.cast<mlir::FloatType>().getFloatSemantics(),
            /*Negative=*/true));
    SmallVector<Value, 4> outTensorShape;
    // `maxpool2d` contains the result of maxpool2d operation over the input.
    Value maxPool2d, paddedInput;
    if (failed(createPoolingOp<linalg::PoolingNchwMaxOp>(
            op, rewriter, self, /*supportNonFPInput=*/false, ceilMode,
            kernelSizeIntValues, strideInts, paddingInts, dilationInts,
            smallestFPValueAttr, outTensorShape, paddedInput, maxPool2d)))
      return rewriter.notifyMatchFailure(op, "unable to compute maxpool2d");
    Type newResultType = getTypeConverter()->convertType(op.getType());
    rewriter.replaceOpWithNewOp<tensor::CastOp>(op, newResultType,
        maxPool2d);
    return success();
  }
};
```

MLIR - Pattern Rewrites

https://github.com/llvm/torch-mlir/blob/main/lib/Conversion/TorchToLinalg/Pooling.cpp#L128

```
class ConvertAtenMaxPool2dOp : public OpConversionPattern<AtenMaxPool2dOp> {
public:
  using OpConversionPattern::OpConversionPattern;
  LogicalResult
  matchAndRewrite(AtenMaxPool2dOp op, OpAdaptor adaptor,
                  ConversionPatternRewriter &rewriter) const override {
    if (failed(verifyLinalgCompatibleTypes(op, rewriter)))
      return failure();

    TypeConverter *typeConverter = getTypeConverter();
    Value self = adaptor.getSelf();
    int64_t selfRank = self.getType().cast<RankedTensorType>().getRank();
    // TODO: Add support for 3D inputs.
    if (selfRank == 3)
      return rewriter.notifyMatchFailure(
          op, "unimplemented: only support 4D input");

    bool ceilMode;
    SmallVector<Value, 2> kernelSizeIntValues;
    SmallVector<int64_t, 2> strideInts, paddingInts, dilationInts;
    if (!matchPattern(op.getDilation(), m_TorchListOfConstantInts(dilationInts)))
      return rewriter.notifyMatchFailure(op,
                                         "only support constant int dilations");
    if (failed(checkAndGetPoolingParameters<AtenMaxPool2dOp>(
            op, rewriter, typeConverter, ceilMode, kernelSizeIntValues,
            strideInts, paddingInts)))
      return rewriter.notifyMatchFailure(op, "invalid pooling parameters");
    Type elementType =
        self.getType().cast<RankedTensorType>().getElementType();
    auto smallestFPValueAttr = rewriter.getFloatAttr(
        elementType,
        APFloat::getLargest(
            elementType.cast<mlir::FloatType>().getFloatSemantics(),
            /*Negative=*/true));
    SmallVector<Value, 4> outTensorShape;
    // `maxpool2d` contains the result of maxpool2d operation over the input.
    Value maxPool2d, paddedInput;
    if (failed(createPoolingOp<linalg::PoolingNchwMaxOp>(
            op, rewriter, self, /*supportNonFPInput=*/false, ceilMode,
            kernelSizeIntValues, strideInts, paddingInts, dilationInts,
            smallestFPValueAttr, outTensorShape, paddedInput, maxPool2d)))
      return rewriter.notifyMatchFailure(op, "unable to compute maxpool2d");
    Type newResultType = getTypeConverter()->convertType(op.getType());
    rewriter.replaceOpWithNewOp<tensor::CastOp>(op, newResultType,
        maxPool2d);
    return success();
  }
};
```

MLIR - Pattern Rewrites

https://github.com/llvm/torch-mlir/blob/main/lib/Conversion/TorchToLinalg/Pooling.cpp#L128

```
class ConvertAtenMaxPool2dOp : public OpConversionPattern<AtenMaxPool2dOp> {
public:
  using OpConversionPattern::OpConversionPattern;
  LogicalResult
  matchAndRewrite(AtenMaxPool2dOp op, OpAdaptor adaptor,
                  ConversionPatternRewriter &rewriter) const override {
    if (failed(verifyLinalgCompatibleTypes(op, rewriter)))
      return failure();

    TypeConverter *typeConverter = getTypeConverter();
    Value self = adaptor.getSelf();
    int64_t selfRank = self.getType().cast<RankedTensorType>().getRank();
    // TODO: Add support for 3D inputs.
    if (selfRank == 3)
      return rewriter.notifyMatchFailure(
          op, "unimplemented: only support 4D input");

    bool ceilMode;
    SmallVector<Value, 2> kernelSizeIntValues;
    SmallVector<int64_t, 2> strideInts, paddingInts, dilationInts;
    if (!matchPattern(op.getDilation(), m_TorchListOfConstantInts(dilationInts)))
      return rewriter.notifyMatchFailure(op,
                                         "only support constant int dilations");
    if (failed(checkAndGetPoolingParameters<AtenMaxPool2dOp>(
            op, rewriter, typeConverter, ceilMode, kernelSizeIntValues,
            strideInts, paddingInts)))
      return rewriter.notifyMatchFailure(op, "invalid pooling parameters");
    Type elementType =
        self.getType().cast<RankedTensorType>().getElementType();
    auto smallestFPValueAttr = rewriter.getFloatAttr(
        elementType,
        APFloat::getLargest(
            elementType.cast<mlir::FloatType>().getFloatSemantics(),
            /*Negative=*/true));
    SmallVector<Value, 4> outTensorShape;
    // `maxpool2d` contains the result of maxpool2d operation over the input.
    Value maxPool2d, paddedInput;
    if (failed(createPoolingOp<linalg::PoolingNchwMaxOp>(
            op, rewriter, self, /*supportNonFPInput=*/false, ceilMode,
            kernelSizeIntValues, strideInts, paddingInts, dilationInts,
            smallestFPValueAttr, outTensorShape, paddedInput, maxPool2d)))
      return rewriter.notifyMatchFailure(op, "unable to compute maxpool2d");
    Type newResultType = getTypeConverter()->convertType(op.getType());
    rewriter.replaceOpWithNewOp<tensor::CastOp>(op, newResultType,
        maxPool2d);
    return success();
  }
};
```

MLIR - Pattern Rewrites

https://github.com/llvm/torch-mlir/blob/main/lib/Conversion/TorchToLinalg/Pooling.cpp#L128

```
class ConvertAtenMaxPool2dOp : public OpConversionPattern<AtenMaxPool2dOp> {
public:
  using OpConversionPattern::OpConversionPattern;
  LogicalResult
  matchAndRewrite(AtenMaxPool2dOp op, OpAdaptor adaptor,
                  ConversionPatternRewriter &rewriter) const override {
    if (failed(verifyLinalgCompatibleTypes(op, rewriter)))
      return failure();

    TypeConverter *typeConverter = getTypeConverter();
    Value self = adaptor.getSelf();
    int64_t selfRank = self.getType().cast<RankedTensorType>().getRank();
    // TODO: Add support for 3D inputs.
    if (selfRank == 3)
      return rewriter.notifyMatchFailure(
          op, "unimplemented: only support 4D input");

    bool ceilMode;
    SmallVector<Value, 2> kernelSizeIntValues;
    SmallVector<int64_t, 2> strideInts, paddingInts, dilationInts;
    if (!matchPattern(op.getDilation(), m_TorchListOfConstantInts(dilationInts)))
      return rewriter.notifyMatchFailure(op,
                                         "only support constant int dilations");
    if (failed(checkAndGetPoolingParameters<AtenMaxPool2dOp>(
            op, rewriter, typeConverter, ceilMode, kernelSizeIntValues,
            strideInts, paddingInts)))
      return rewriter.notifyMatchFailure(op, "invalid pooling parameters");
    Type elementType =
        self.getType().cast<RankedTensorType>().getElementType();
    auto smallestFPValueAttr = rewriter.getFloatAttr(
        elementType,
        APFloat::getLargest(
            elementType.cast<mlir::FloatType>().getFloatSemantics(),
            /*Negative=*/true));
    SmallVector<Value, 4> outTensorShape;
    // `maxpool2d` contains the result of maxpool2d operation over the input.
    Value maxPool2d, paddedInput;
    if (failed(createPoolingOp<linalg::PoolingNchwMaxOp>(
            op, rewriter, self, /*supportNonFPInput=*/false, ceilMode,
            kernelSizeIntValues, strideInts, paddingInts, dilationInts,
            smallestFPValueAttr, outTensorShape, paddedInput, maxPool2d)))
      return rewriter.notifyMatchFailure(op, "unable to compute maxpool2d");
    Type newResultType = getTypeConverter()->convertType(op.getType());
    rewriter.replaceOpWithNewOp<tensor::CastOp>(op, newResultType,
        maxPool2d);
    return success();
  }
};
```

MLIR - Pass Infrastructure

https://mlir.llvm.org/docs/PassManagement

- Pass Managers
  - `OperationPass` covers everything: Operation, Block, Region, Function, and Module
- Passes group sets of Transformation Patterns
  - Conversion
    - Dialect A -> Dialect B
    - Convert Types and Operations
    - Full or Partial
    - E.g., “Tensor” -> “MemRef”
  - Canonicalization
    - Dialect A -> Dialect A
    - E.g., “x+x -> x*2”, Fusion, Tiling, etc.
- Analyses
  - No common base class, constructed on the fly
  - Passes must register analysis state invalidation

MLIR - PDL

https://github.com/llvm/llvm-project/blob/main/mlir/test/Rewrite/pdl-bytecode.mlir

- "traditional" Pattern Rewrites in C++
  - But also possible declaratively:

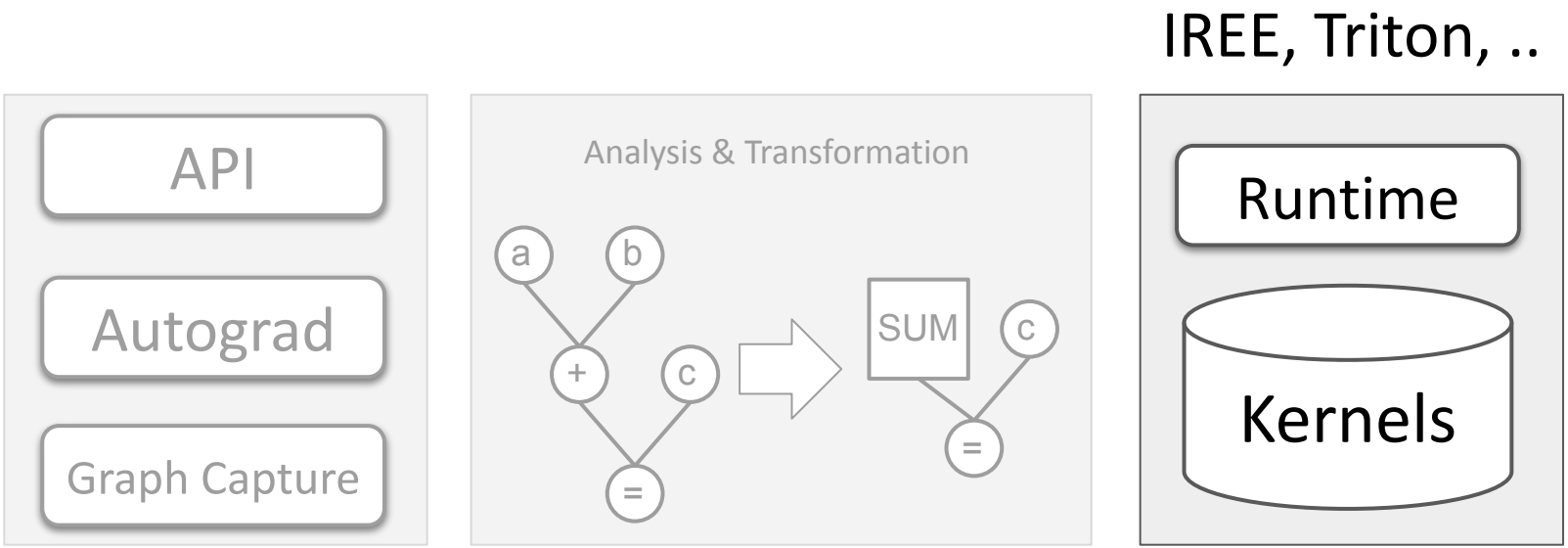
```
def : Pat<
  (AOp $input, ignored_attr),
  (DOp (BOp:$b_result) $b_result)
>;
```

- Generalization: DSL for Pattern Rewrites
  - https://mlir.llvm.org/docs/PDLL/
  - Transform MLIR using MLIR!
  - Dialect "PDL Interp"
  - Has Bytecode definition and Interpreter

```
module @patterns {
  pdl_interp.func @matcher(%root : !pdl.operation) {
    %results = pdl_interp.get_results of %root : !pdl.range<value>
    %types = pdl_interp.get_value_type of %results : !pdl.range<type>
    pdl_interp.apply_constraint
"multi_entity_var_constraint"(%results, %types : !pdl.range<value>,
!pdl.range<type>) -> ^pat, ^end

  ^pat:
    pdl_interp.record_match @rewriters::@success(%root :
!pdl.operation) : benefit(1), loc([%root]) -> ^end

  ^end:
    pdl_interp.finalize
  }

  module @rewriters {
    pdl_interp.func @success(%root : !pdl.operation) {
      %op = pdl_interp.create_operation "test.replaced_by_pattern"
      pdl_interp.erase %root
      pdl_interp.finalize
    }
  }
}
```

But wait! There's more: 2020 LLVM Developers' Meeting: M. Amini & R. Riddle "MLIR Tutorial"
https://www.youtube.com/watch?v=Y4SvqTtOIDk

Deep Learning Compilers



Q&A

Thank you!