

Scaling PyTorch Model Training With Minimal Code Changes

 @rasbt

 sebastian@lightning.ai

 <https://sebastianraschka.com>



Lightning ^{AI}

<https://lightning.ai>

**It's 2023, training from scratch often
doesn't make sense anymore**

Training a vision transformer from scratch

```
Epoch: 0001/0010 | Batch 0000/2812 | Loss: 2.5971
Epoch: 0001/0010 | Batch 0300/2812 | Loss: 1.9497
Epoch: 0001/0010 | Batch 0600/2812 | Loss: 1.6241
...
Epoch: 0001/0010 | Batch 2700/2812 | Loss: 1.5693
Epoch: 0001/0010 | Train acc.: 34.81% | Val acc.: 47.22%
Epoch: 0002/0010 | Batch 0000/2812 | Loss: 1.0045
...
Epoch: 0004/0010 | Batch 2700/2812 | Loss: 1.1540
Epoch: 0004/0010 | Train acc.: 59.33% | Val acc.: 59.14%
...
Epoch: 0005/0010 | Train acc.: 62.04% | Val acc.: 60.06%
...
Epoch: 0006/0010 | Train acc.: 64.22% | Val acc.: 61.80%
...
Epoch: 0007/0010 | Train acc.: 66.12% | Val acc.: 61.14%
Epoch: 0008/0010 | Batch 0000/2812 | Loss: 0.5717
...
Epoch: 0010/0010 | Batch 2700/2812 | Loss: 0.6933
Epoch: 0010/0010 | Train acc.: 71.16% | Val acc.: 62.80%
Time elapsed 61.48 min
Test accuracy 62.85%
```

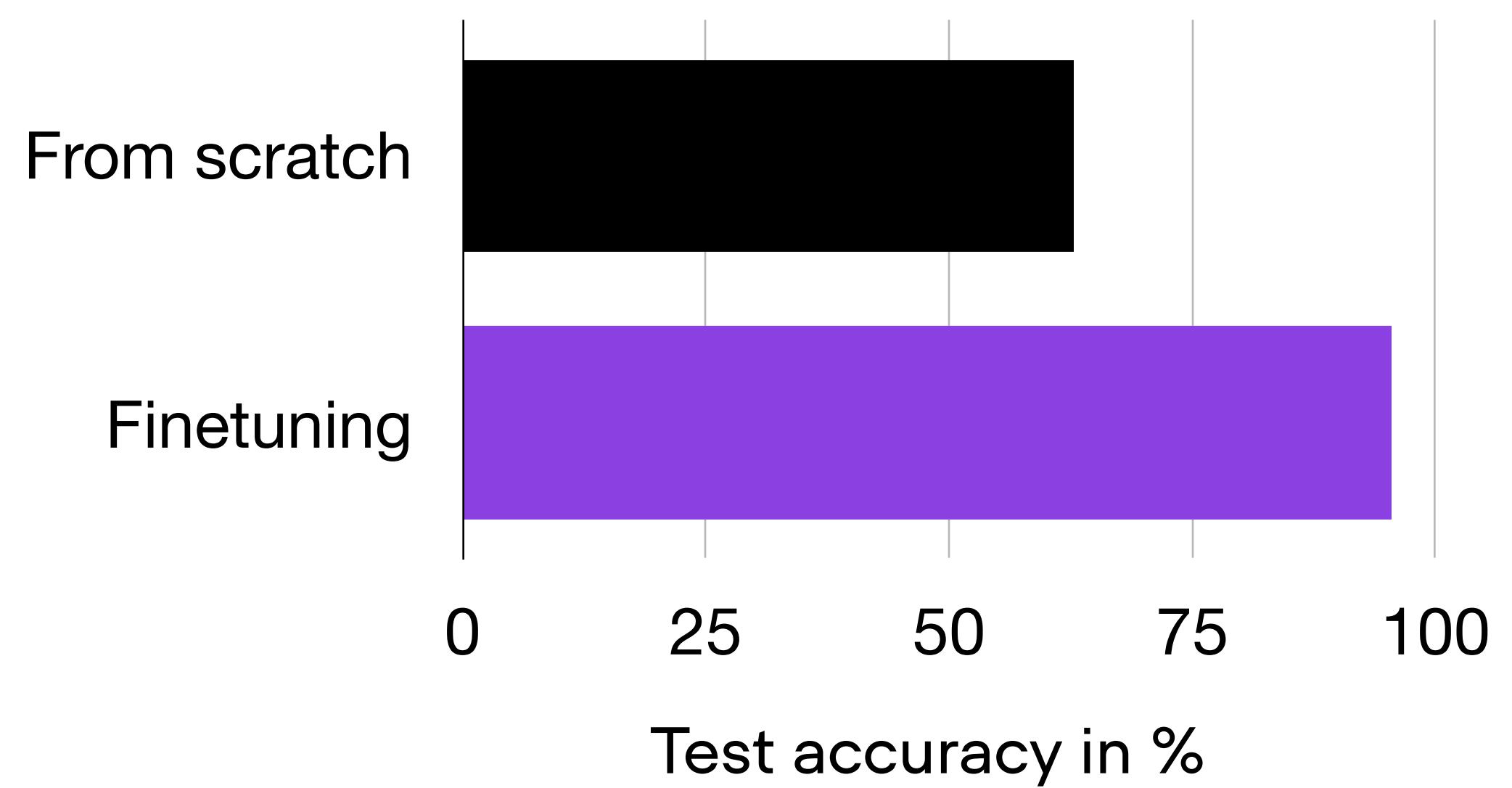
Training a vision transformer from scratch

```
Epoch: 0001/0010 | Batch 0000/2812 | Loss: 2.5971
Epoch: 0001/0010 | Batch 0300/2812 | Loss: 1.9497
Epoch: 0001/0010 | Batch 0600/2812 | Loss: 1.6241
...
Epoch: 0001/0010 | Batch 2700/2812 | Loss: 1.5693
Epoch: 0001/0010 | Train acc.: 34.81% | Val acc.: 47.22%
Epoch: 0002/0010 | Batch 0000/2812 | Loss: 1.0045
...
Epoch: 0004/0010 | Batch 2700/2812 | Loss: 1.1540
Epoch: 0004/0010 | Train acc.: 59.33% | Val acc.: 59.14%
...
Epoch: 0005/0010 | Train acc.: 62.04% | Val acc.: 60.06%
...
Epoch: 0006/0010 | Train acc.: 64.22% | Val acc.: 61.80%
...
Epoch: 0007/0010 | Train acc.: 66.12% | Val acc.: 61.14%
Epoch: 0008/0010 | Batch 0000/2812 | Loss: 0.5717
...
Epoch: 0010/0010 | Batch 2700/2812 | Loss: 0.6933
Epoch: 0010/0010 | Train acc.: 71.16% | Val acc.: 62.80%
Time elapsed 61.48 min
Test accuracy 62.85%
```

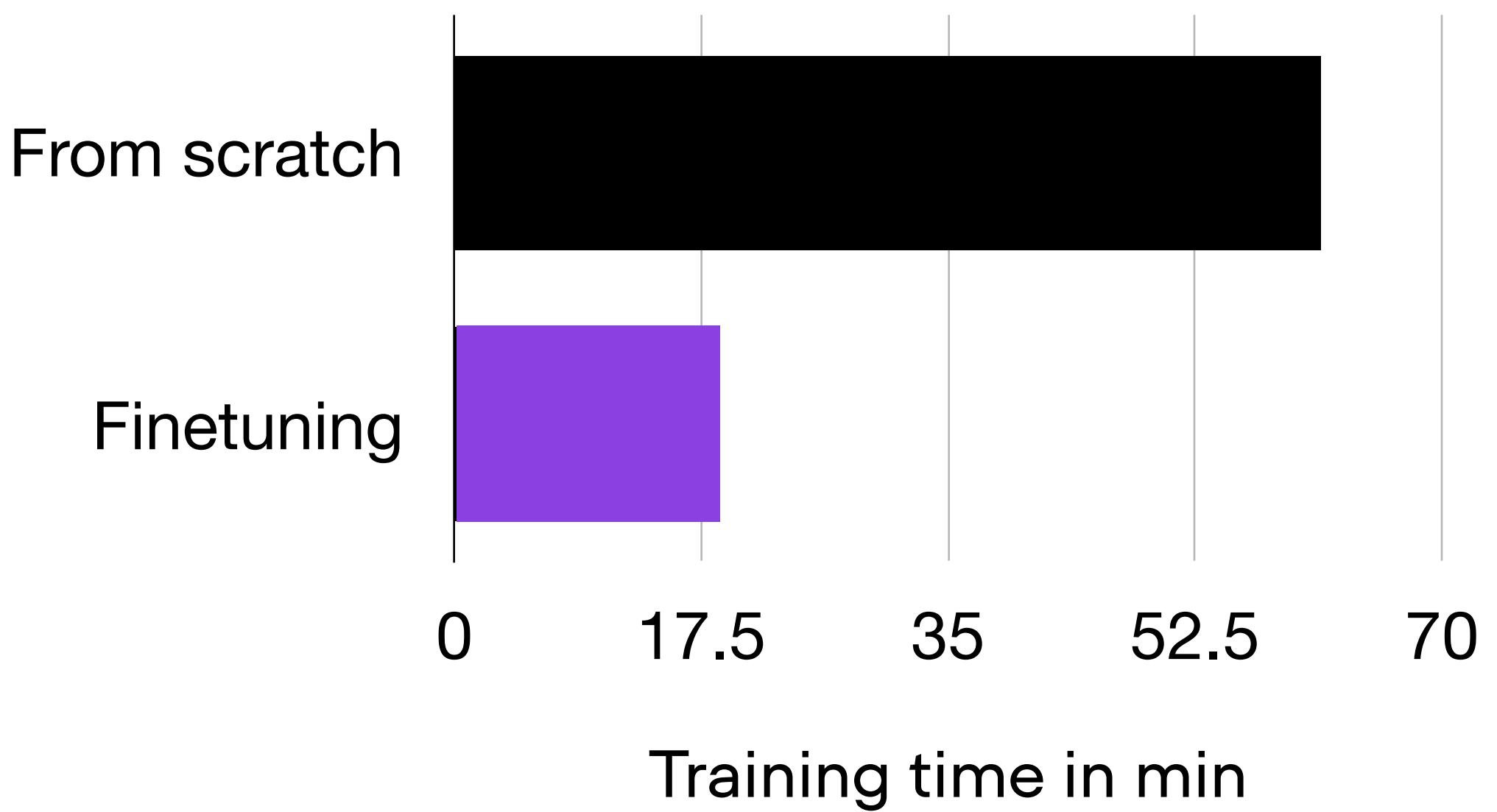
Finetuning a pretrained vision transformer

```
Epoch: 0001/0003 | Batch 0000/2812 | Loss: 2.4934
Epoch: 0001/0003 | Batch 0300/2812 | Loss: 0.0954
Epoch: 0001/0003 | Batch 0600/2812 | Loss: 0.0981
Epoch: 0001/0003 | Batch 0900/2812 | Loss: 0.2078
Epoch: 0001/0003 | Batch 1200/2812 | Loss: 0.3588
Epoch: 0001/0003 | Batch 1500/2812 | Loss: 0.0104
Epoch: 0001/0003 | Batch 1800/2812 | Loss: 0.1560
Epoch: 0001/0003 | Batch 2100/2812 | Loss: 0.0474
Epoch: 0001/0003 | Batch 2400/2812 | Loss: 0.4250
Epoch: 0001/0003 | Batch 2700/2812 | Loss: 0.4414
Epoch: 0001/0003 | Train acc.: 92.40% | Val acc.: 94.12%
Epoch: 0002/0003 | Batch 0000/2812 | Loss: 0.0912
Epoch: 0002/0003 | Batch 0300/2812 | Loss: 0.0337
Epoch: 0002/0003 | Batch 0600/2812 | Loss: 0.1545
Epoch: 0002/0003 | Batch 0900/2812 | Loss: 0.0478
...
Epoch: 0003/0003 | Batch 2100/2812 | Loss: 0.0060
Epoch: 0003/0003 | Batch 2400/2812 | Loss: 0.1395
Epoch: 0003/0003 | Batch 2700/2812 | Loss: 0.1128
Epoch: 0003/0003 | Train acc.: 97.24% | Val acc.: 95.74%
Time elapsed 18.70 min
Test accuracy 95.37%
```

Accuracy (larger is better)



Time to convergence (lower is better)





<https://github.com/rasbt/cvpr2023>

How can we accelerate the training even further?

Fabric is the fast and lightweight way to scale
PyTorch models without boilerplate code

Suppose we have the following plain PyTorch code

```
● ● ●  
1 import torch  
2 import torch.nn as nn  
3 from torch.utils.data import DataLoader, Dataset  
4  
5  
6 class PyTorchModel(nn.Module):  
7     # ...  
8  
9  
10    class PyTorchDataset(Dataset):  
11        # ...  
12  
13  
14        num_epochs = 10  
15        loss_fn = torch.nn.functional.cross_entropy()  
16  
17        device = "cuda" if torch.cuda.is_available() else "cpu"  
18        model = PyTorchModel(...)  
19        optimizer = torch.optim.SGD(model.parameters())  
20  
21        dataloader = DataLoader(PyTorchDataset(...), ...)  
22  
23        model.train()  
24  
25        for epoch in range(num_epochs):  
26            for batch in dataloader:  
27                input, target = input.to(device), target.to(device)  
28                optimizer.zero_grad()  
29                output = model(input)  
30                loss = loss_fn(output, target)  
31                loss.backward()  
32                optimizer.step()  
33  
34
```

**Fabric* brings advanced functionality without
restructuring**

**Fabric* brings advanced functionality without
restructuring**

* 100% free an open source

PyTorch

```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader, Dataset
4
5-
6
7 class PyTorchModel(nn.Module):
8     # ...
9
10 class PyTorchDataset(Dataset):
11     # ...
12
13 loss_fn = torch.nn.functional.cross_entropy()
14 device = "cuda" if torch.cuda.is_available() else "cpu"
15 model = PyTorchModel(...)
16 optimizer = torch.optim.SGD(model.parameters())
17 dataloader = DataLoader(PyTorchDataset(...), ...)
18
19 model.train()
20
21 for epoch in range(num_epochs):
22     for batch in dataloader:
23         input, target = input.to(device), target.to(device)
24         optimizer.zero_grad()
25         output = model(input)
26         loss = loss_fn(output, target)
27         loss.backward()
28         optimizer.step()
```

PyTorch + Fabric

```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader, Dataset
4
5+ from lightning.fabric import Fabric
6
7 class PyTorchModel(nn.Module):
8     # ...
9
10 class PyTorchDataset(Dataset):
11     # ...
12+
13+ fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
14+ fabric.launch()
15+
16 loss_fn = torch.nn.functional.cross_entropy()
17 model = PyTorchModel(...)
18 optimizer = torch.optim.SGD(model.parameters())
19+ model, optimizer = fabric.setup(model, optimizer)
20 dataloader = DataLoader(PyTorchDataset(...), ...)
21+ dataloader = fabric.setup_dataloaders(dataloader)
22 model.train()
23
24 for epoch in range(num_epochs):
25     for batch in dataloader:
26+         input, target = batch
27         optimizer.zero_grad()
28         output = model(input)
29         loss = loss_fn(output, target)
30+         fabric.backward(loss)
31         optimizer.step()
```

Convert PyTorch to Fabric Step by Step

```
● ● ●  
1 import torch  
2 import torch.nn as nn  
3 from torch.utils.data import DataLoader, Dataset  
4  
5 from lightning.fabric import Fabric  
6  
7 class PyTorchModel(nn.Module):  
8     # ...  
9  
10    class PyTorchDataset(Dataset):  
11        # ...  
12  
13    fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")  
14    fabric.launch()  
15  
16    loss_fn = torch.nn.functional.cross_entropy()  
17    model = PyTorchModel(...)  
18    optimizer = torch.optim.SGD(model.parameters())  
19    dataloader = DataLoader(PyTorchDataset(...), ...)  
20  
21    model, optimizer = fabric.setup(model, optimizer)  
22    dataloader = fabric.setup_dataloaders(dataloader)  
23  
24    model.train()  
25  
26    for epoch in range(num_epochs):  
27        for batch in dataloader:  
28            input, target = batch  
29            optimizer.zero_grad()  
30            output = model(input)  
31            loss = loss_fn(output, target)  
32            fabric.backward(loss)  
33            optimizer.step()  
34
```



```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader, Dataset
4
5 from lightning.fabric import Fabric
6
7 class PyTorchModel(nn.Module):
8     # ...
9
10 class PyTorchDataset(Dataset):
11     # ...
12
13 fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
14 fabric.launch()
15
16 loss_fn = torch.nn.functional.cross_entropy()
17 model = PyTorchModel(...)
18 optimizer = torch.optim.SGD(model.parameters())
19 dataloader = DataLoader(PyTorchDataset(...), ...)
20
21 model, optimizer = fabric.setup(model, optimizer)
22 dataloader = fabric.setup_dataloaders(dataloader)
23
24 model.train()
25
26 for epoch in range(num_epochs):
27     for batch in dataloader:
28         input, target = batch
29         optimizer.zero_grad()
30         output = model(input)
31         loss = loss_fn(output, target)
32         fabric.backward(loss)
33         optimizer.step()
34
```

Import, instantiate, and launch Fabric



```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader, Dataset
4
5 from lightning.fabric import Fabric
6
7 class PyTorchModel(nn.Module):
8     # ...
9
10 class PyTorchDataset(Dataset):
11     # ...
12
13 fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
14 fabric.launch()
15
16 loss_fn = torch.nn.functional.cross_entropy()
17 model = PyTorchModel(...)
18 optimizer = torch.optim.SGD(model.parameters())
19 dataloader = DataLoader(PyTorchDataset(...), ...)
20
21 model, optimizer = fabric.setup(model, optimizer)
22 dataloader = fabric.setup_dataloaders(dataloader)
23
24 model.train()
25
26 for epoch in range(num_epochs):
27     for batch in dataloader:
28         input, target = batch
29         optimizer.zero_grad()
30         output = model(input)
31         loss = loss_fn(output, target)
32         fabric.backward(loss)
33         optimizer.step()
34
```

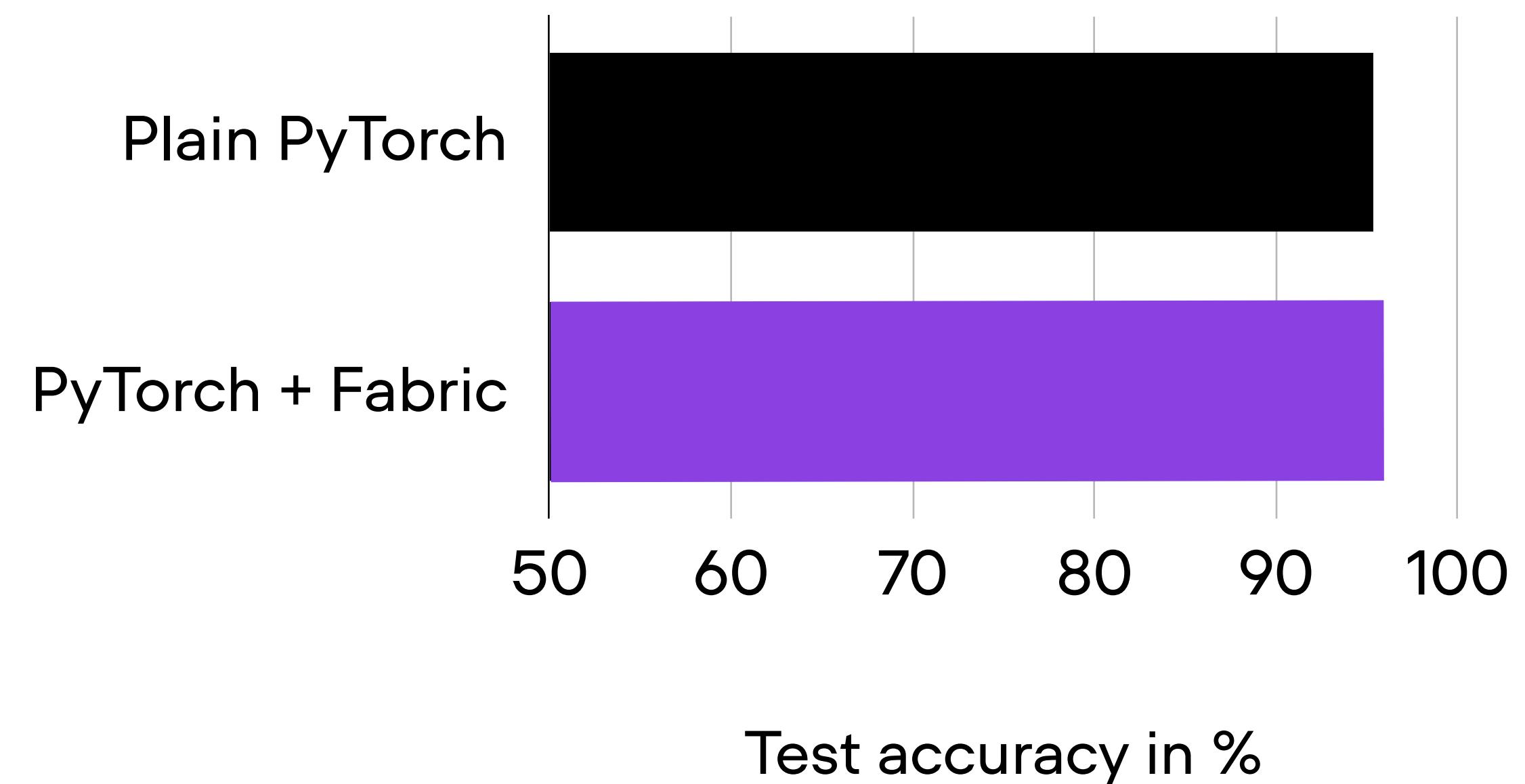
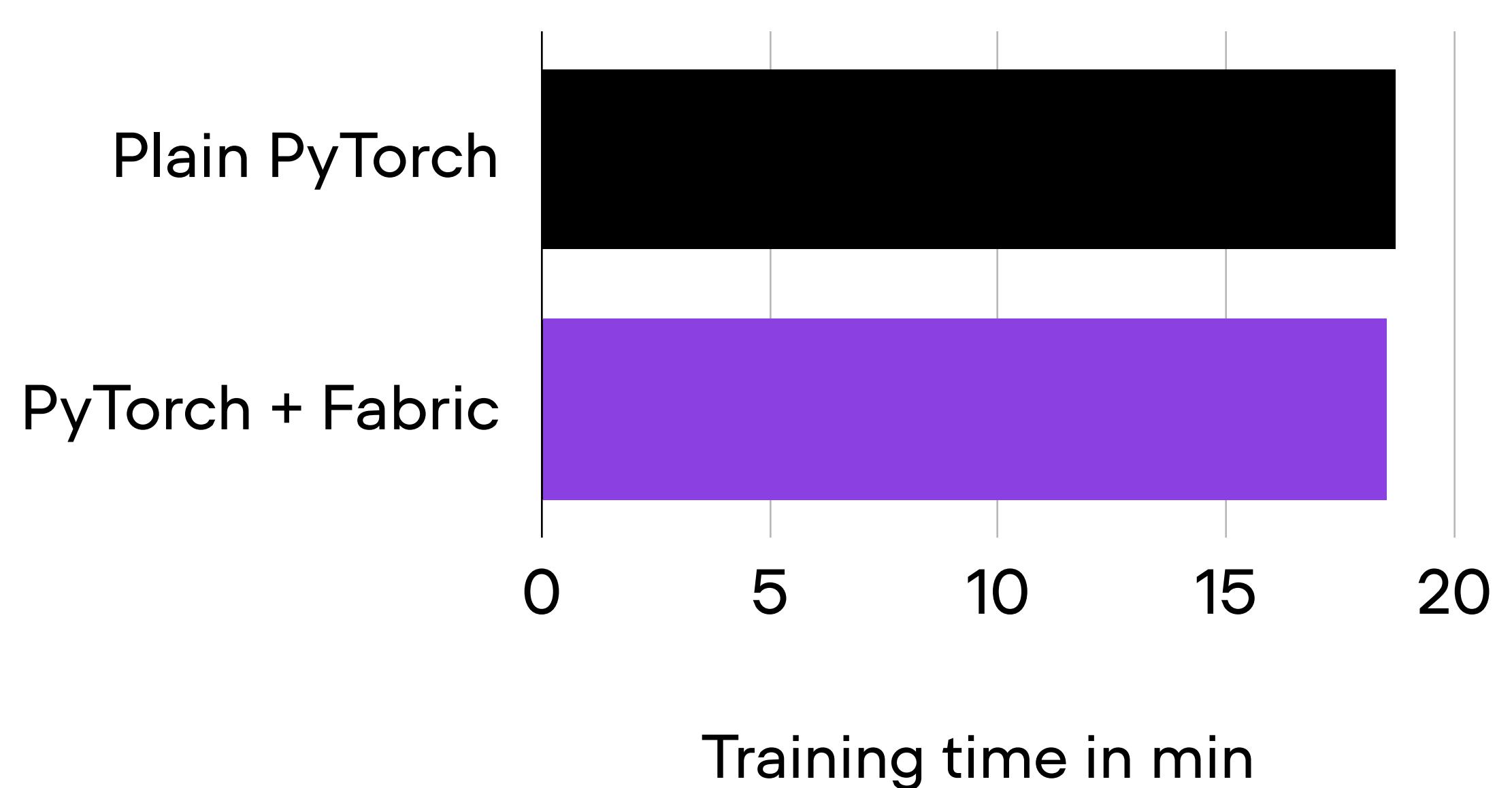
Set up model, optimizer, and data loader



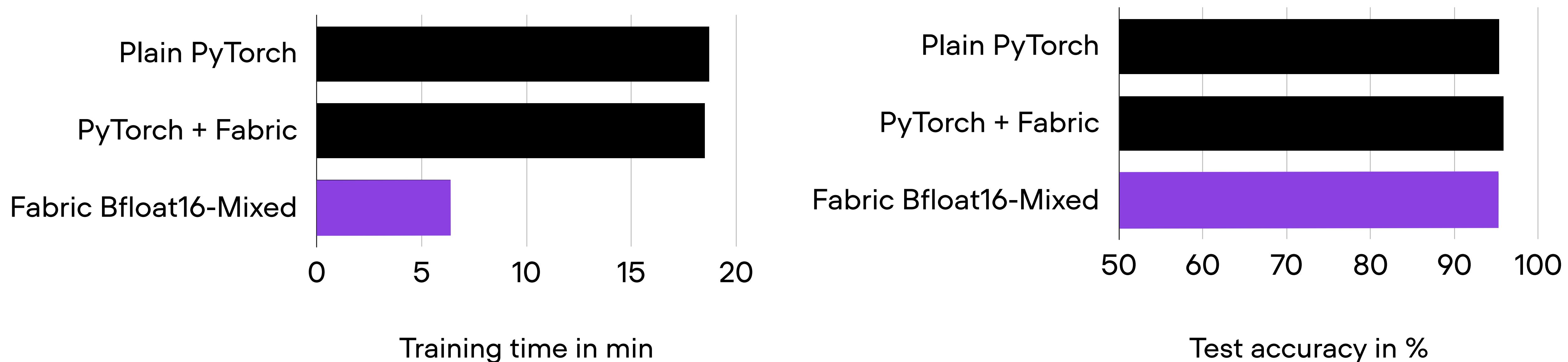
```
1 import torch
2 import torch.nn as nn
3 from torch.utils.data import DataLoader, Dataset
4
5 from lightning.fabric import Fabric
6
7 class PyTorchModel(nn.Module):
8     # ...
9
10 class PyTorchDataset(Dataset):
11     # ...
12
13 fabric = Fabric(accelerator="cuda", devices=8, strategy="ddp")
14 fabric.launch()
15
16 loss_fn = torch.nn.functional.cross_entropy()
17 model = PyTorchModel(...)
18 optimizer = torch.optim.SGD(model.parameters())
19 dataloader = DataLoader(PyTorchDataset(...), ...)
20
21 model, optimizer = fabric.setup(model, optimizer)
22 dataloader = fabric.setup_dataloaders(dataloader)
23
24 model.train()
25
26 for epoch in range(num_epochs):
27     for batch in dataloader:
28         input, target = batch
29         optimizer.zero_grad()
30         output = model(input)
31         loss = loss_fn(output, target)
32         fabric.backward(loss) # Call fabric.backward
33         optimizer.step()
34
```

Use `fabric.backward(loss)`

A Performance Baseline



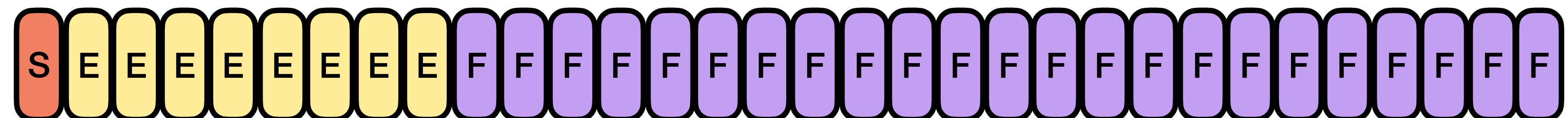
Using Mixed-Precision Training



Required Change: 1/2 Lines of Code

```
fabric = Fabric(accelerator="cuda", devices=1, precision="bf16-mixed")
```

Float 32

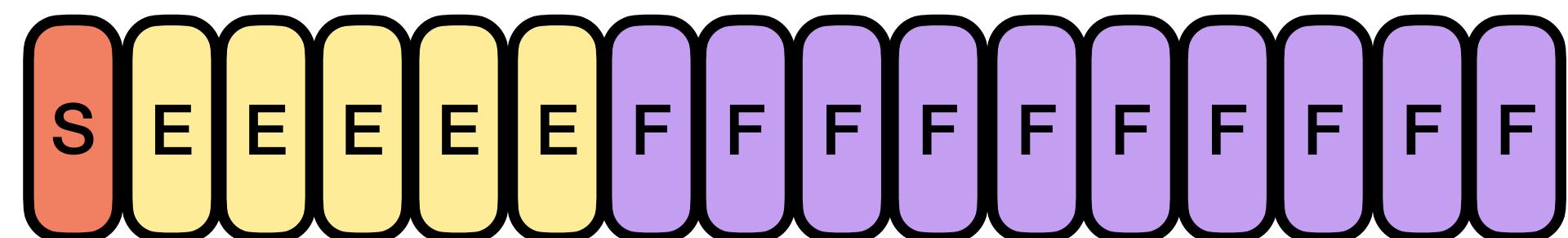


Sign
(1 bit)

Exponent
(8 bits)

Fraction
(23 bits)

Float 16 ("half" precision)

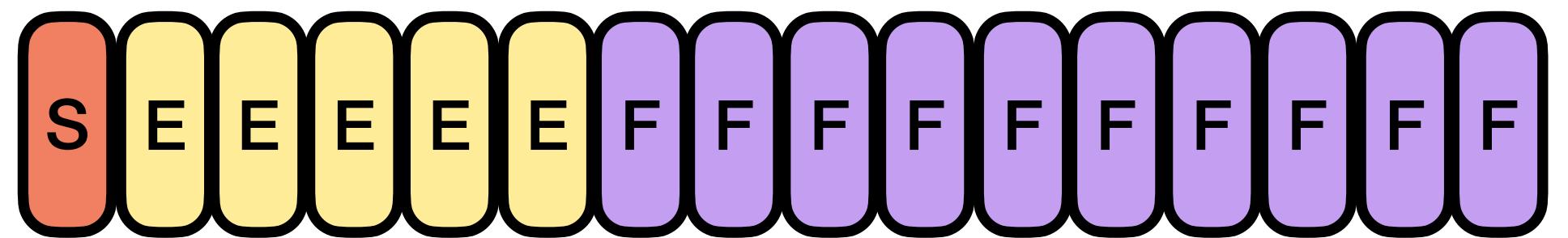


Sign
(1 bit)

Exponent
(5 bits)

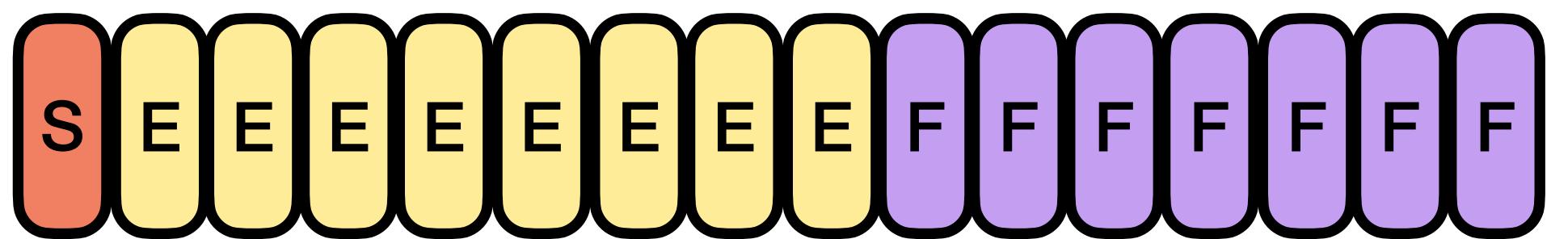
Fraction
(10 bits)

Float 16 ("half" precision)



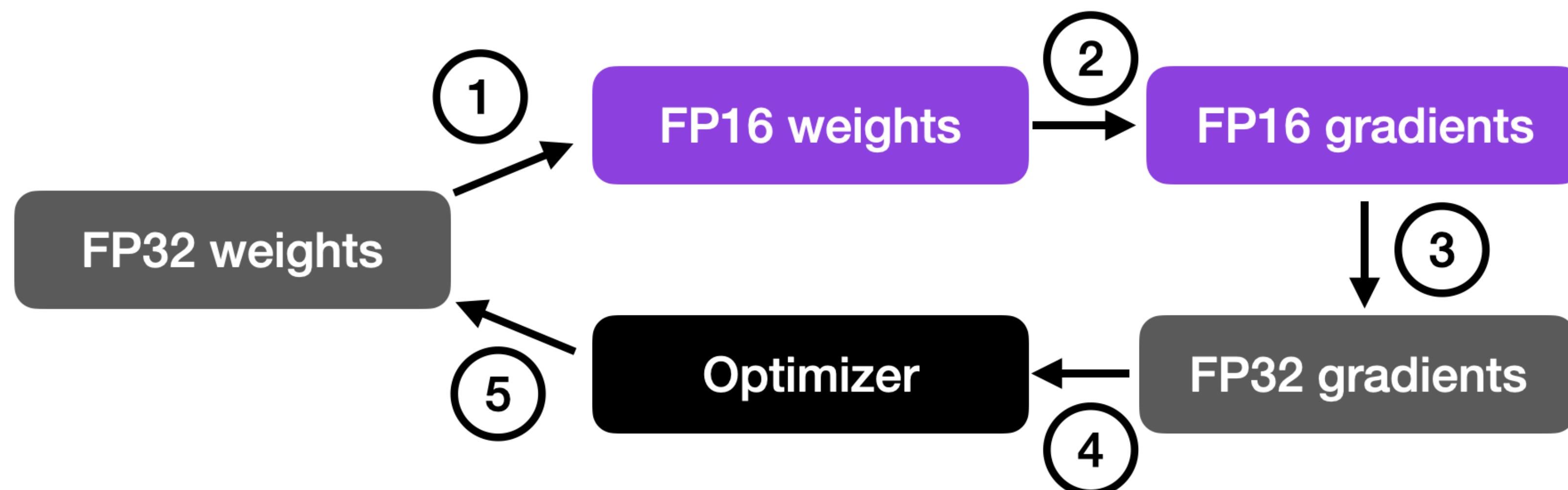
Sign (1 bit)	Exponent (5 bits)	Fraction (10 bits)
-----------------	----------------------	-----------------------

Bfloat 16 ("brain" floating point, more "dynamic range" like float 32)



Sign (1 bit)	Exponent (8 bits)	Fraction (7 bits)
-----------------	----------------------	----------------------

Using Mixed-Precision Training



Multi-GPU Training with Fully Sharded Data Parallelism

Broad Categories of Parallelism for Multi-GPU Training

Model parallelism

Data parallelism

Pipeline parallelism

Tensor parallelism

Sequence parallelism

Broad Categories of Parallelism for Multi-GPU Training

Model parallelism

Data parallelism

Pipeline parallelism

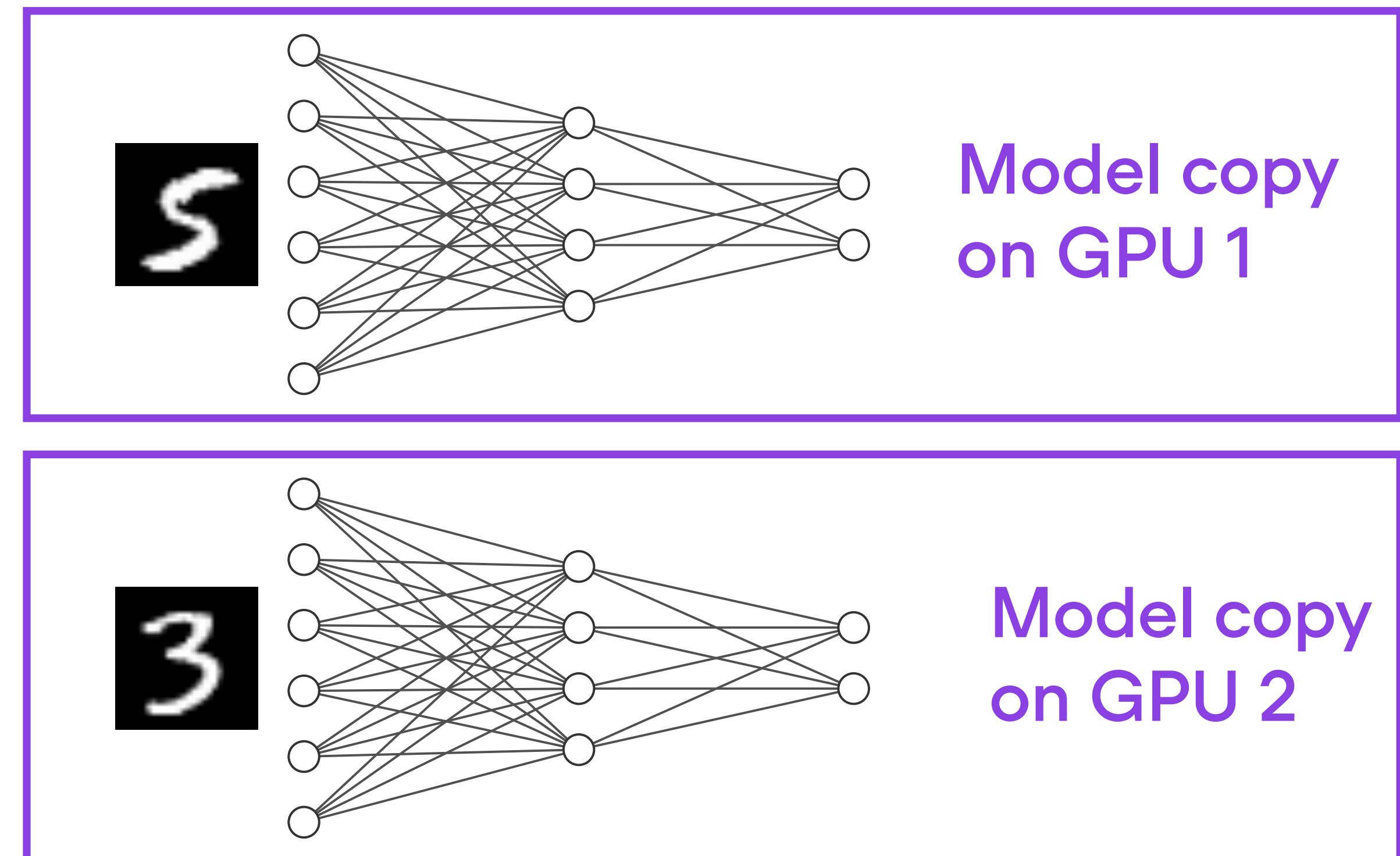
Tensor parallelism

Sequence parallelism



Data Parallelism

**Split batch to train
model(s) on more data
in parallel**

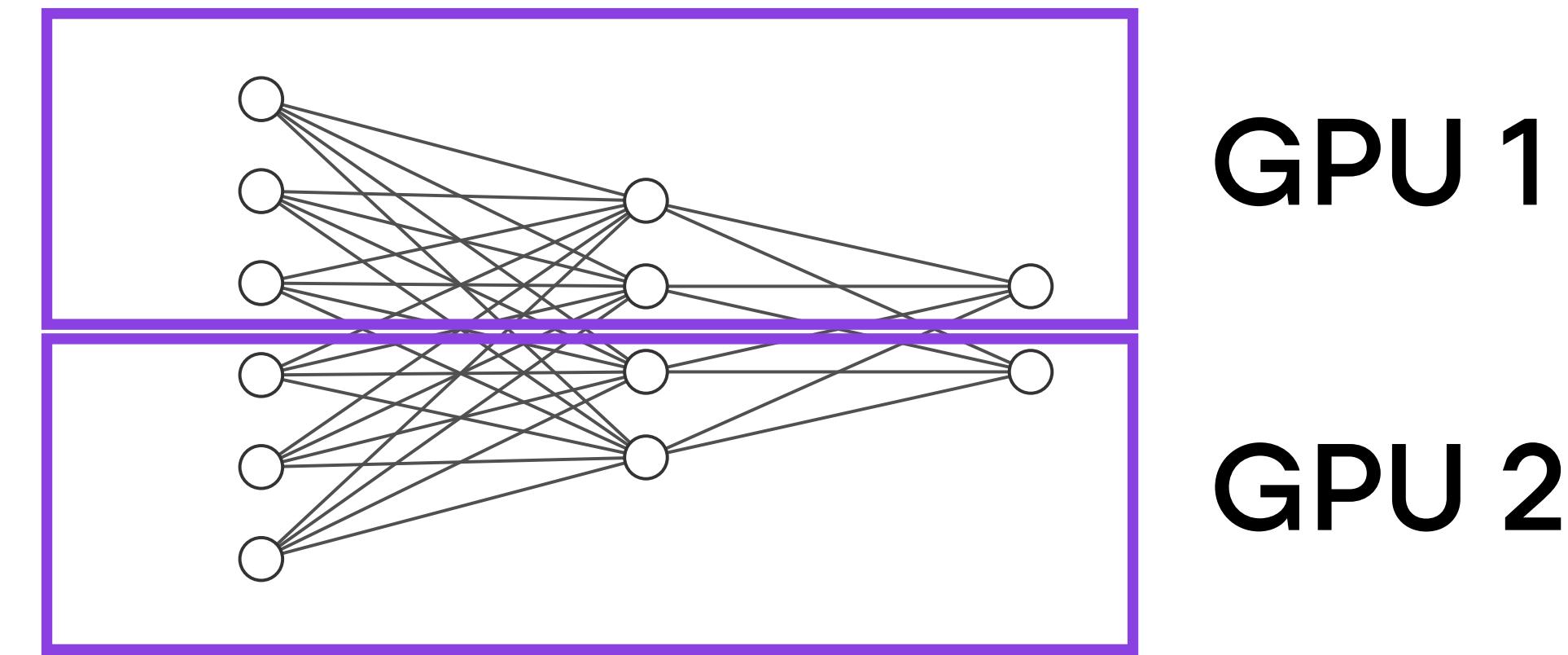


**Model copy
on GPU 1**

**Model copy
on GPU 2**

Tensor Parallelism

Related to model parallelism, but split horizontally instead of vertically



(Depending on the implementation, you may split weight matrices, optimizer states, gradients)

Tensor Parallelism

For example, split the matrix multiplication **by column**

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{matrix} \cdot \begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} = \begin{matrix} 22 & 28 \\ 49 & 64 \end{matrix}$$

Tensor Parallelism

For example, split the matrix multiplication **by column**

The diagram illustrates the decomposition of a matrix multiplication into three parallel operations based on column splitting.

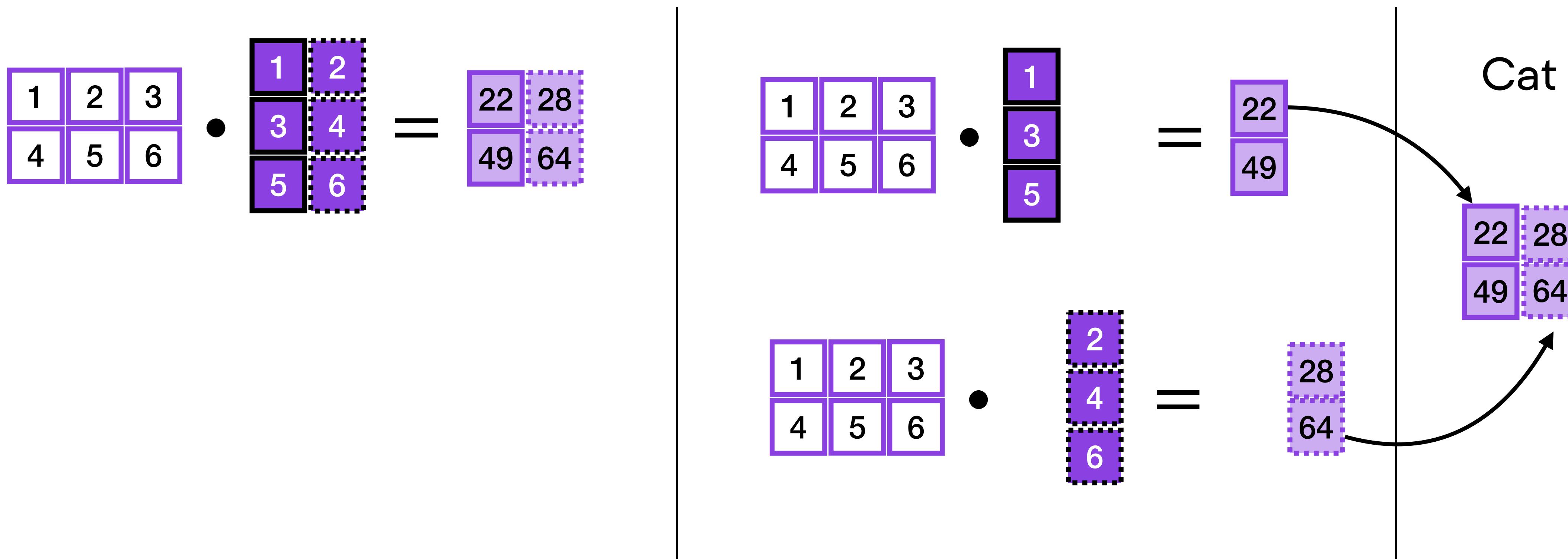
Left Operation: A 2x3 matrix (1, 2, 3; 4, 5, 6) is multiplied by a 3x2 matrix (1, 2; 3, 4; 5, 6). The result is a 2x2 matrix (22, 28; 49, 64). The second column of the first matrix and the first column of the second matrix are highlighted with dashed boxes.

Middle Operation: The same 2x3 matrix is multiplied by a 3x1 matrix (1; 3; 5). The result is a 2x1 matrix (22; 49).

Right Operation: The same 2x3 matrix is multiplied by a 3x1 matrix (2; 4; 6). The result is a 2x1 matrix (28; 64).

Tensor Parallelism

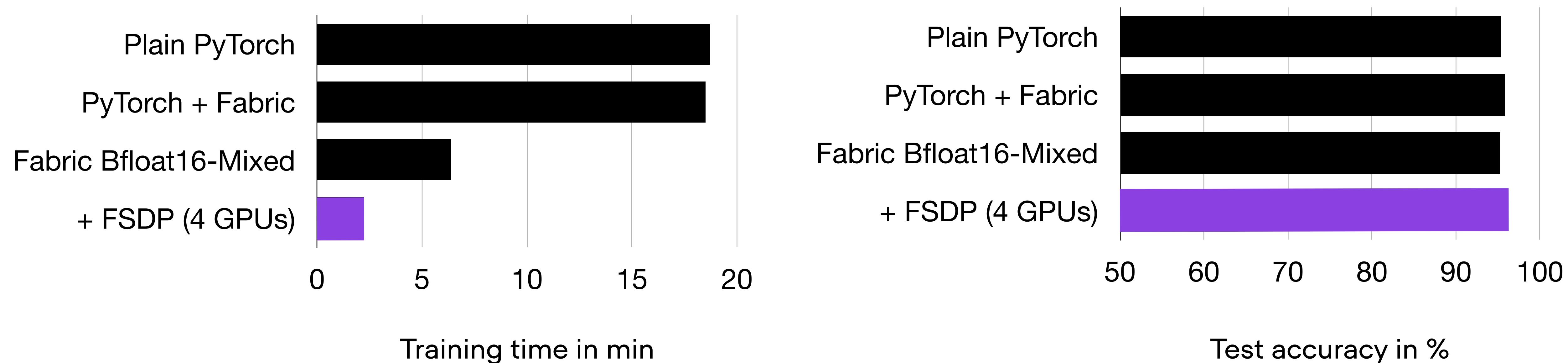
For example, split the matrix multiplication **by column**



Required Change: 1/2 Lines of Code

```
fabric = Fabric(  
    accelerator="cuda", precision="bf16-mixed",  
    devices=4, strategy="FSDP"  
)
```

Multi-GPU Training with Fully Sharded Data Parallelism



Contact

 @rasbt

 sebastian@lightning.ai

 <https://sebastianraschka.com>

 <https://lightning.ai>

Code & slides

 <https://github.com/rasbt/cvpr2023>