Implementation

CODING TIME!

Imports

```
1 import os
2 import torch
3 import torch.distributed as dist
4 import torch.nn as nn
5 import torch.optim as optim
6 from torch.nn.parallel import DistributedData...
7 from torchvision import datasets, transforms,...
8 from torch.utils.data import DataLoader,...
```

These are standard PyTorch and torchvision libraries for:

- Building and training models
- Distributed training
- Using datasets and transformations

Distributed Setup

```
def setup():
    dist.init_process_group(
        backend='gloo',
        init_method='env://',
        world_size=int(os.environ['WORLD_SIZE']),
        rank=int(os.environ['RANK'].split('-')[-1])
    )
```

Explanation:

- Initializes the DDP process group.
- `backend='gloo'` for CPU or simple GPU communication.
- `init_method='env://'` uses environment variables.
- Extracts `rank` from strings like `"ddp-trainer-0"`.

Distributed Setup

```
def setup():
    dist.init_process_group(
        backend='gloo',
        init_method='env://',
        world_size=int(os.environ['WORLD_SIZE']),
        rank=int(os.environ['RANK'].split('-')[-1])
    )
```

Explanation:

- Initializes the DDP process group.
- `backend='gloo'` for CPU or simple GPU communication.
- `init_method='env://'` uses environment variables.
- Extracts `rank` from strings like `"ddp-trainer-0"`.

Cleanup

```
1 def cleanup():
2     dist.destroy_process_group()
```

Releases resources and cleanly exits the process group after training.

Main Function (Start)

```
1 def main():
2     setup()
```

Begins by initializing the distributed training environment.

Dataset and Transform

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

dataset = datasets.CIFAR10(root='/srv/nfs/cifar10'...
```

Explanation:

- Converts images to tensors and normalizes them.
- CIFAR-10 dataset is downloaded to a shared NFS directory.

Distributed Sampler and DataLoader

```
sampler = DistributedSampler(dataset)
dataloader = DataLoader(dataset, sampler=sampler...
```

Explanation:

- `DistributedSampler` splits the dataset across processes.
- `DataLoader` uses the sampler to prevent data duplication.

Model and DDP Wrapping

```
model = models.resnet18(num_classes=10)
model = DDP(model)
```

Explanation:

- Loads ResNet18 with 10 output classes.
- Wraps the model with `DistributedDataParallel` for synced training.

Loss Function and Optimizer

```
loss_fn = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Explanation:

- Uses cross-entropy for classification.
- Optimizes using Adam with learning rate 0.001.

Training Loop

```
for epoch in range(3):
    print("epoch", epoch)
    ctr = 0
    for images, labels in dataloader:
        ctr += 1
        print("hello123", ctr)
        outputs = model(images)
        loss = loss_fn(outputs, labels)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        if ctr % 10 == 0:
            print(f"[RANK␣{os.environ['RANK']}]...
␣␣␣␣print(f"[RANK {os.environ['RANK']}]...
```

Explanation:

- Trains for 3 epochs, printing progress every 10 steps.

Final Cleanup and Entry Point

```
1 cleanup()
2
3 if __name__ == "__main__":
4     main()
```

Explanation:

- Cleans up the distributed environment at the end.
- Ensures `main()` only runs when script is executed directly.