**FSDP Production Readiness**

**FullyShardedDataParallel (FSDP)**

**What is FSDP?**
PyTorch FullyShardedDataParallel (FSDP) implements data parallelism as an `nn.Module` wrapper while sharing parameters, gradients, and optimizer states across workers to enable large models. In other words, FSDP preserves the simplicity of data parallel training without requiring the model to fit on one GPU.

**How Does FSDP Work?**
FSDP modifies the communication schedule of vanilla data parallel training:
- Before forward or backward computation, FSDP unshards the parameters with an all-gather and reshards the parameters after to free memory.
- After backward computation, FSDP synchronizes gradients with a reduce-scatter to have each worker only store a sharded gradient.

**Mixed Precision**
```python
mixed_precision = MixedPrecision(
    param_dtype=torch.bfloat16,   # computation in BF16
    reduce_dtype=torch.bfloat16,  # gradient reduction in BF16
    buffer_dtype=torch.float32,   # buffers stay in FP32
)
```

```
fsdp_model = FSDP(model, mixed_precision=mixed_precision, ...)
```

**Model and Optimizer State Checkpointing**
```python
# Optional: set_state_dict_type(fsdp_model, state_dict_type,
state_dict_config) for different variations
state_dict = fsdp_model.state_dict()
optim_state_dict = FSDP.optim_state_dict(fsdp_model, optim)
fsdp_model.load_state_dict(state_dict)
FSDP.load_optim_state_dict(optim_state_dict, fsdp_model, optim)
```

**Meta Device Initialization**
```python
model = deferred_init(Model, *model_args, **model_kwargs)
fsdp_model = FSDP(model, ...)
```

**Activation Checkpointing**
```python
module_classes: Set[Type[nn.Module]] = ...  # classes to wrap
auto_wrap_policy = ModuleWrapPolicy(module_classes)
fsdp_model = FSDP(model, auto_wrap_policy=auto_wrap_policy, ...)
apply_activation_checkpointing(
    fsdp_model, 
    check_fn=lambda m: isinstance(m, tuple(module_types)),
)
```

**Features for Production Readiness**

**Mixed Precision**

- Native mixed precision
- Model and optimizer state checkpointing
- Interoperability with activation checkpointing
- Meta device initialization
- Memory and throughput optimizations for scaling

**Mixed Precision**
```python
mixed_precision = MixedPrecision(
    param_dtype=torch.bfloat16,   # computation in BF16
    reduce_dtype=torch.bfloat16,  # gradient reduction in BF16
    buffer_dtype=torch.float32,   # buffers stay in FP32
)
```

```
fsdp_model = FSDP(model, mixed_precision=mixed_precision, ...)
```

**Results**

**FSDP with AC on FLAVA**

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>FSDP</th>
<th>FSDP + AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.8B</td>
<td>32.72</td>
<td>63.83 (1.95x)</td>
</tr>
<tr>
<td>2.7B</td>
<td>24.41</td>
<td>55.96 (2.29x)</td>
</tr>
<tr>
<td>4.8B</td>
<td>OOM</td>
<td>49.98</td>
</tr>
<tr>
<td>10B</td>
<td>OOM</td>
<td>22.42</td>
</tr>
</tbody>
</table>

**Mixed Precision on XLMR-700M and DeepViT-8B**

- XLMR-700M: 2 node / 16 40 GB A100 GPUs; batch size 1024
- DeepViT-8B: 1 node / 8 40 GB A100 GPUs

**Meta Device Initialization on GPT-15B**

<table>
<thead>
<tr>
<th>Limiter Off</th>
<th>Limiter On</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Init</td>
<td>1.96 (904x)</td>
</tr>
<tr>
<td>Deferred Init</td>
<td>1.96 (904x)</td>
</tr>
</tbody>
</table>

**Rate Limiter on T5-11B**

<table>
<thead>
<tr>
<th>Batch Size</th>
<th>FSDP</th>
<th>FSDP + AC</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7B</td>
<td>24.41</td>
<td>55.96 (2.29x)</td>
</tr>
<tr>
<td>4.8B</td>
<td>OOM</td>
<td>49.98</td>
</tr>
<tr>
<td>10B</td>
<td>OOM</td>
<td>22.42</td>
</tr>
</tbody>
</table>

**Links**
- [FSDP](https://pytorch.org/docs/stable/fsdp.html)
- [Introduction to PyTorch Fully-Sharded Data Parallel API](https://pytorch.org/blog/introducing-pytorch-fully-sharded-data-parallel-api/)
- [FSDP Tutorial](https://pytorch.org/tutorials/intermediate/FSDP_tutorial.html)

**Acknowledgements**
Yanli Zhao, Anjali Sridhar, Shen Li, Chien-Chin Huang, Less Wright, Bernard Nguyen