TUTORIAL

LLMOPS & MLOPS

Shagun Sodhani
Tech Lead
Meta

Scaling Deep Learning Training with Fully Sharded Data Parallelism in PyTorch

ODSC WEST

OCT 29-31
SAN FRANCISCO

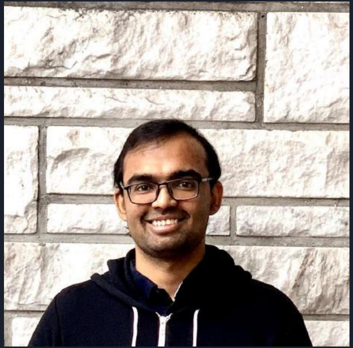
Scaling Deep Learning Training with FSDP in PyTorch

ODSC 2024

@shagunsodhani

About Me

1. Tech Lead and Staff Research Engineer @ Meta AI



2. Focused on building AI agents that can:
    a. interact with and learn from the physical world
    b. consistently improve as they do so without forgetting the previous knowledge

Agenda

1. What is scaling and why care about it
   a. Challenges in scaling deep learning models

2. Fully Sharded Data Parallelism (FSDP)
   a. Overview
   b. FSDP in action

What is scaling

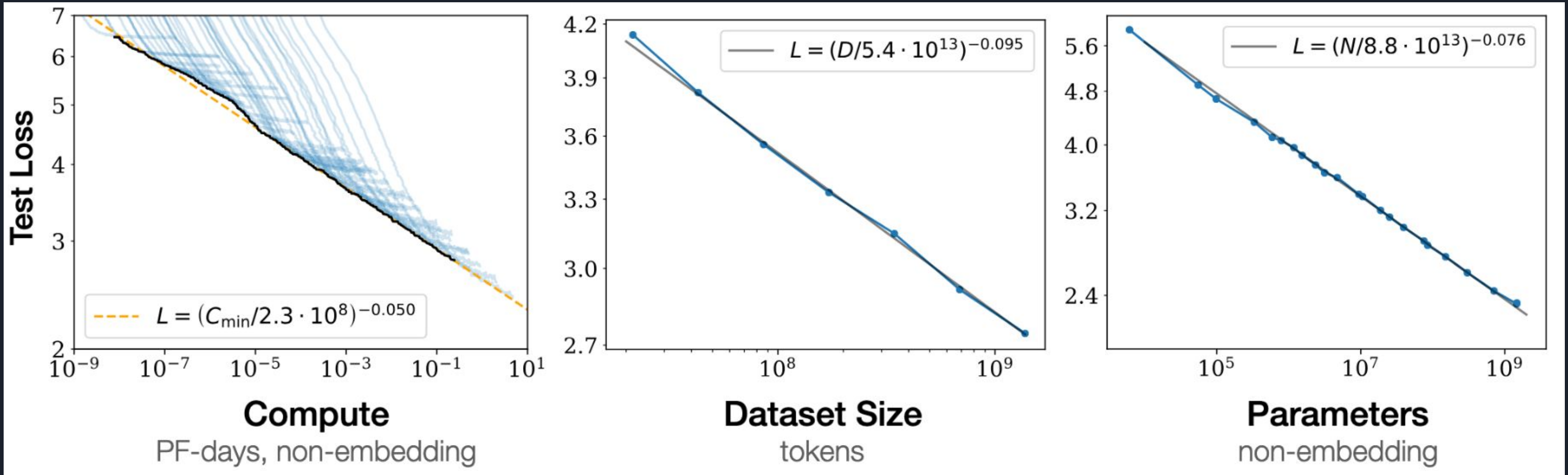
Scaling - Training larger models on larger datasets using more compute

Scaling Hypothesis - Scaling improves performance across diverse tasks.

Strong Scaling Hypothesis - Easiest way to optimize for all the tasks & data is to find a scalable architecture and simply train ever larger NNs [1]

Why care about scaling

Why care about scaling



Taken from [Scaling Laws for Neural Language Models [2]]

Challenges in scaling deep learning models

1. Data Availability and Quality
2. Compute Resource Requirements
3. Cost of Training and Deployment
4. Long Experimentation Cycles
5. Energy Consumption and Environmental Impact
6. Talent and Expertise Gaps
7. …

Challenges when training large models

1. Memory bottlenecks
   a. Size of the model parameters + activations + optimizer state

2. Computation Efficiency
   a. Parallelism overhead can reduce the expected speedups

3. Communication Overhead in Multi-Node Setups
   a. Communication (e.g. for parameter update) limits scalability

Overview of FSDP

Fully Sharded Data Parallelism

Overview of FSDP

Data Parallelism

Overview of FSDP

Data Parallelism

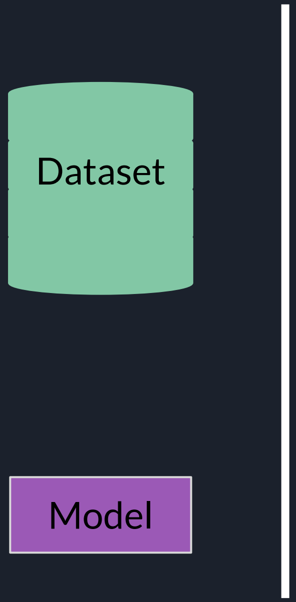
Overview of FSDP

Data Parallelism

Overview of FSDP

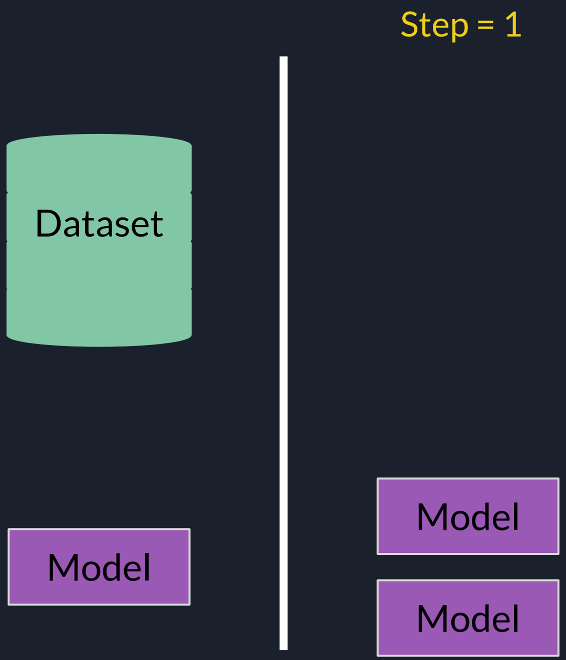
Data Parallelism

Step = 1



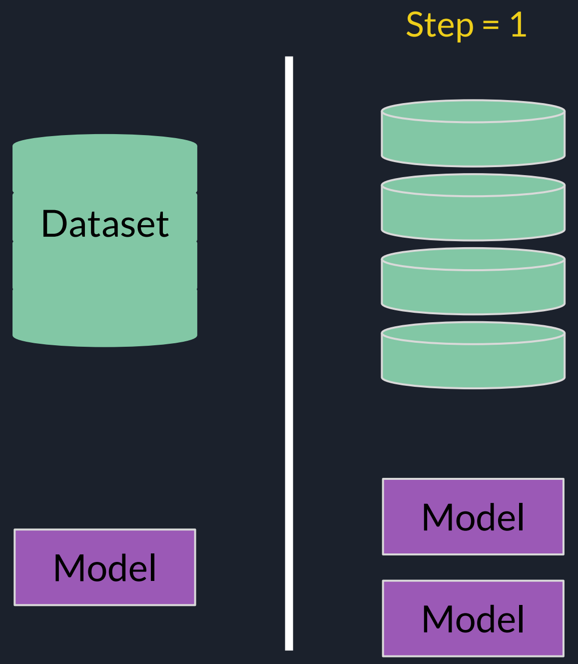
Overview of FSDP

Data Parallelism



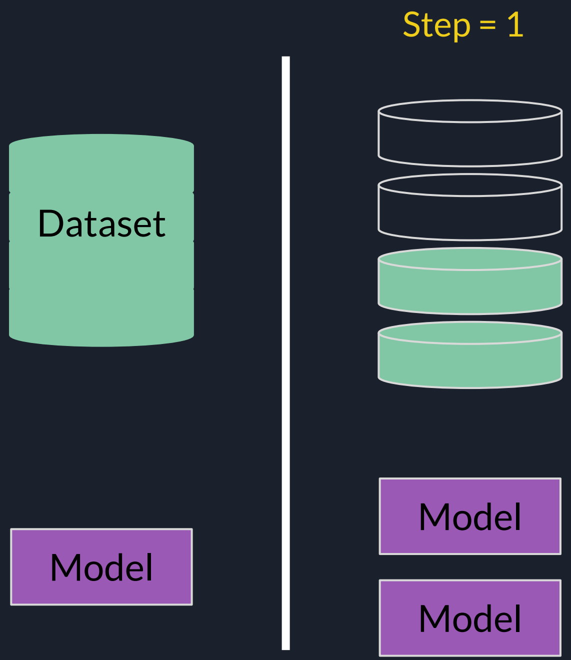
Overview of FSDP

Data Parallelism



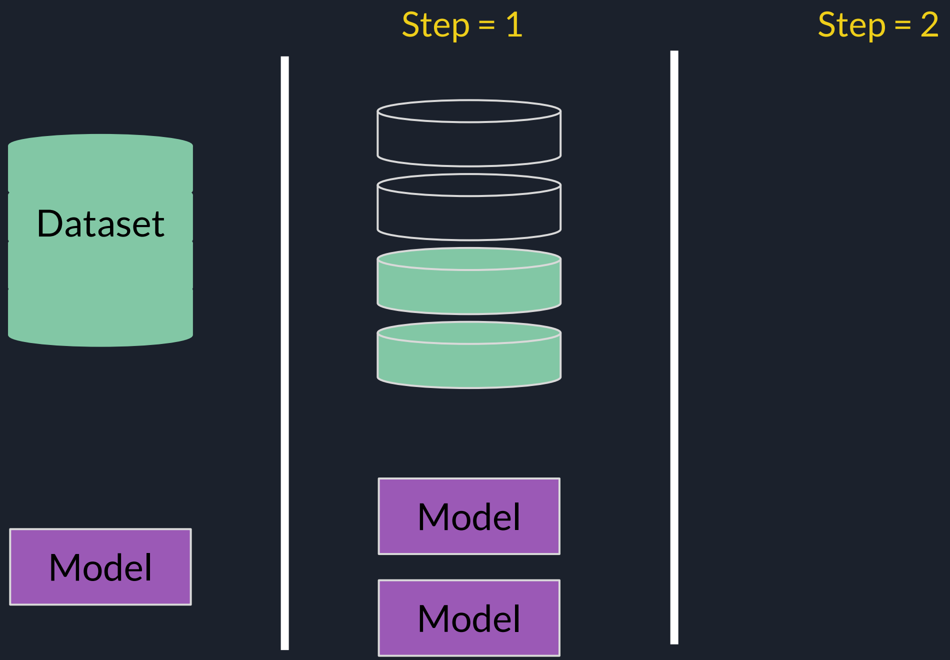
Overview of FSDP

Data Parallelism



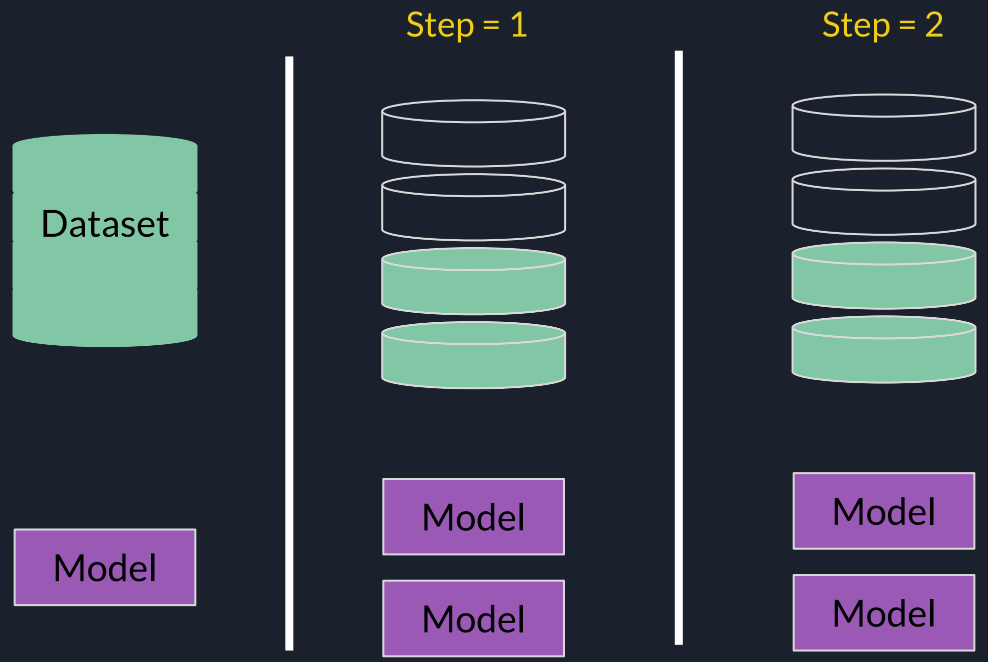
Overview of FSDP

Data Parallelism



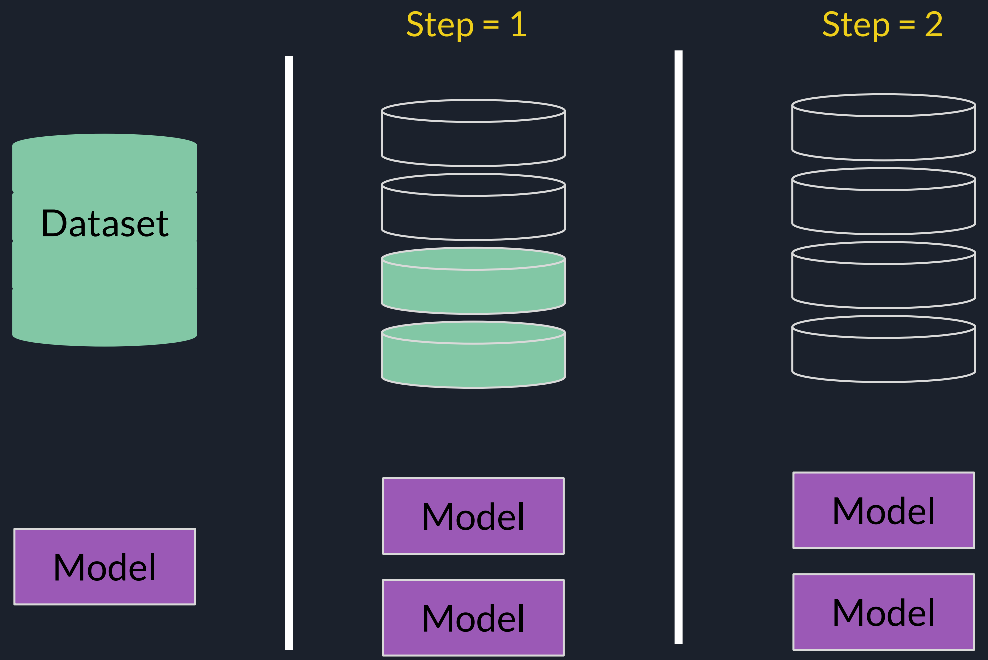
Overview of FSDP

Data Parallelism



Overview of FSDP

Data Parallelism



Overview of FSDP

Data Parallelism

Each GPU has a full copy of the model

Split the dataset in batches and each gpu processes a different batch

Easy to use via *DistributedDataParallel*

Bottlenecked on the size of the model (or activations or optimizer state)

Inefficient for large models or lot of gpus

Overview of FSDP

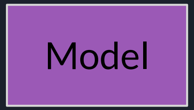
Fully Sharded Data Parallelism

Overview of FSDP

Fully Sharded

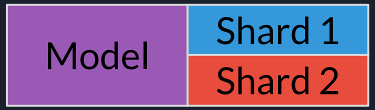
Overview of FSDP

Sharded



Overview of FSDP

Sharded



Overview of FSDP

Sharded

Overview of FSDP

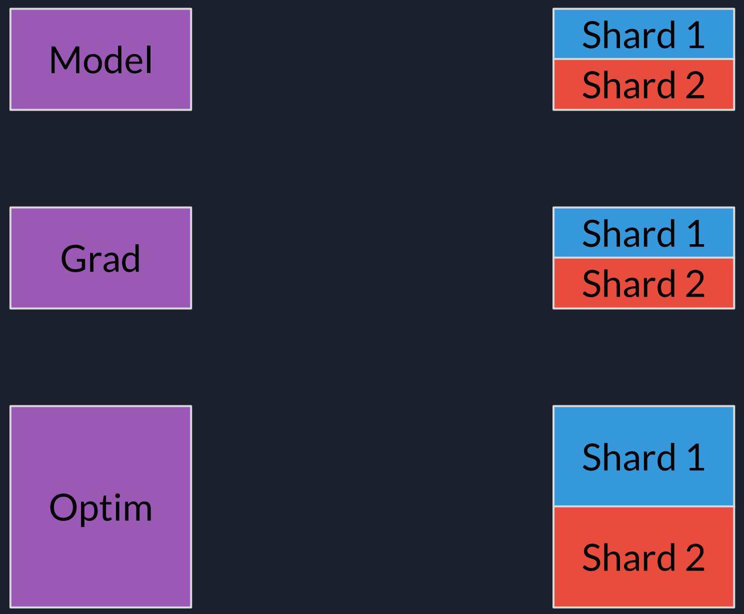
Fully Sharded

Overview of FSDP

Fully Sharded

Overview of FSDP

Fully Sharded

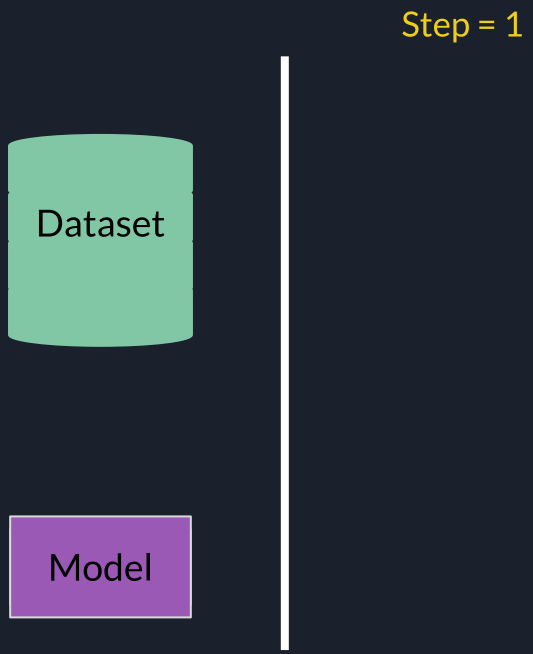


Overview of FSDP

Fully Sharded Data Parallelism

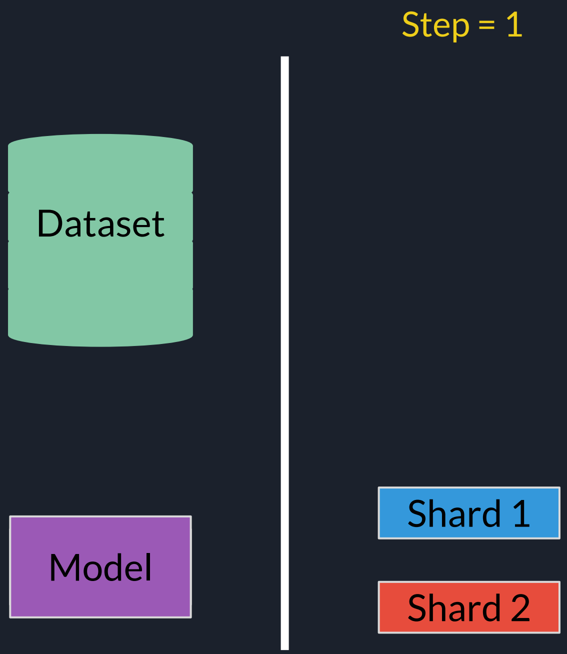
Overview of FSDP

Fully Sharded Data Parallelism



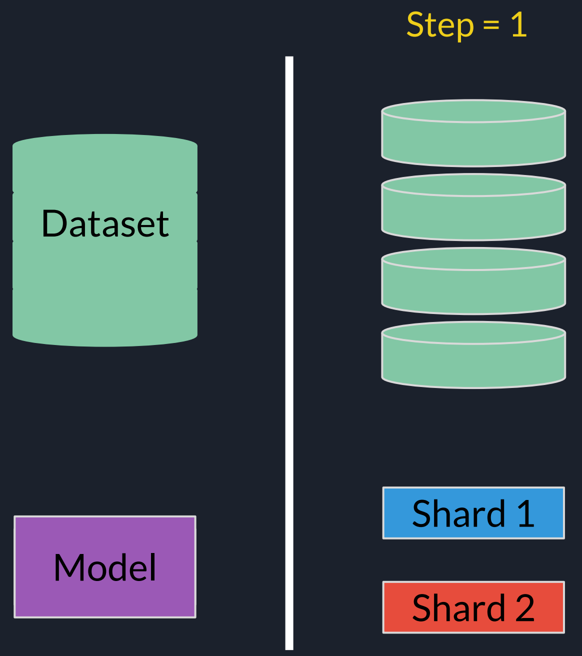
Overview of FSDP

Fully Sharded Data Parallelism



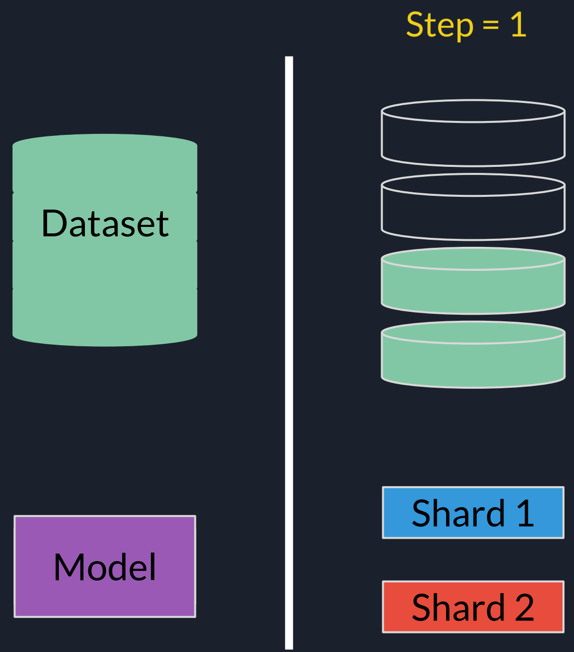
Overview of FSDP

Fully Sharded Data Parallelism



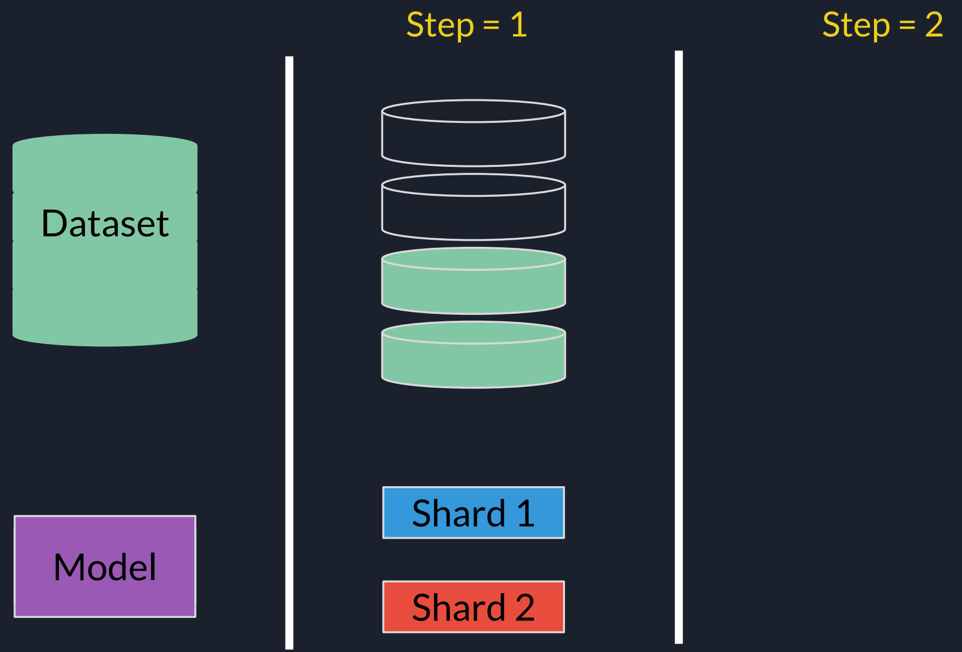
Overview of FSDP

Fully Sharded Data Parallelism



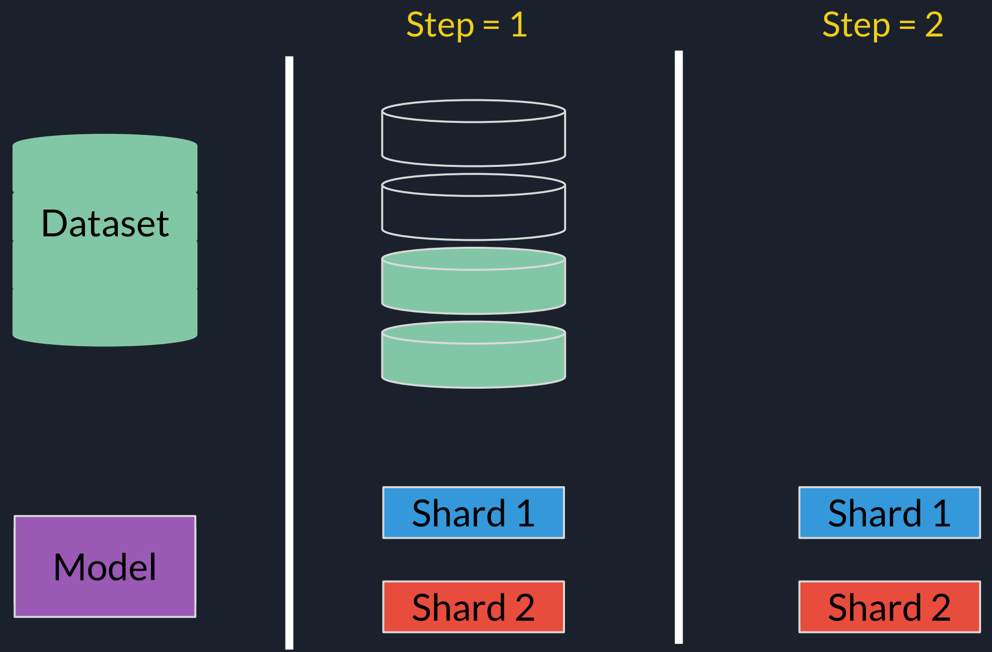
Overview of FSDP

Fully Sharded Data Parallelism



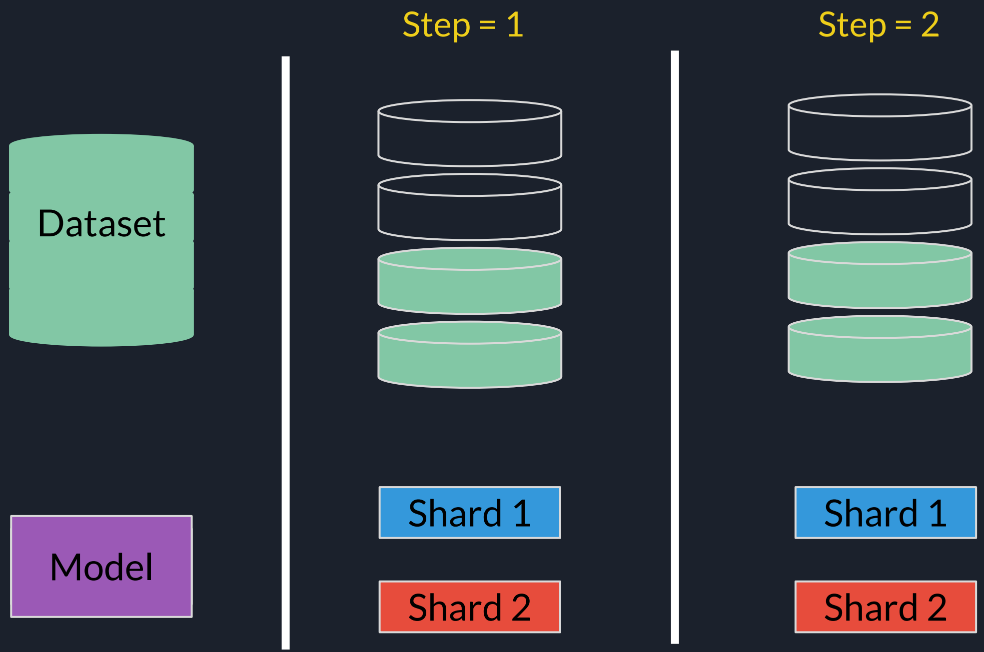
Overview of FSDP

Fully Sharded Data Parallelism



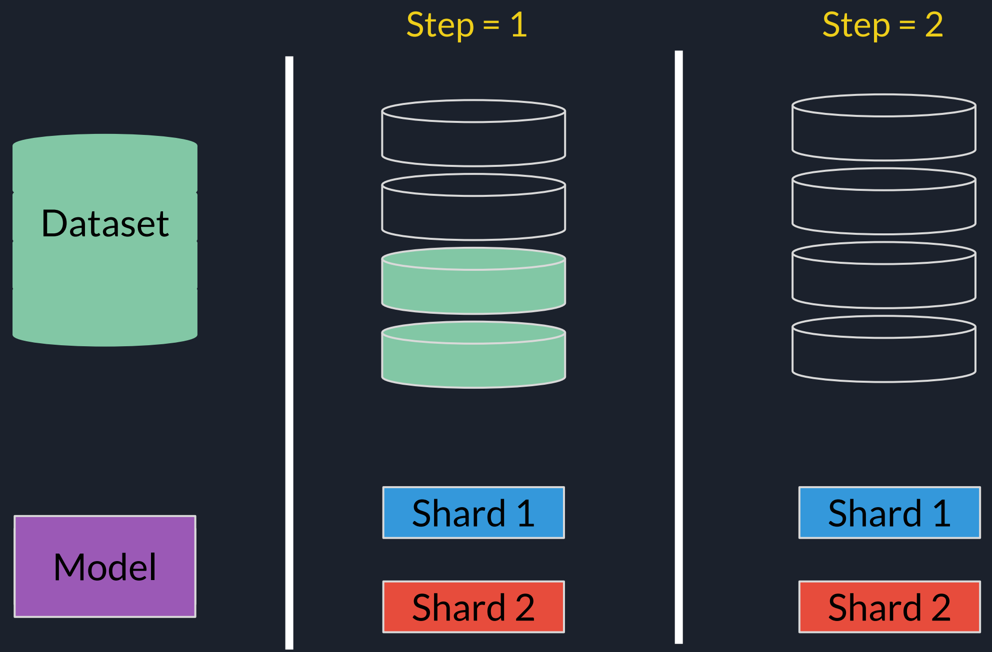
Overview of FSDP

Fully Sharded Data Parallelism

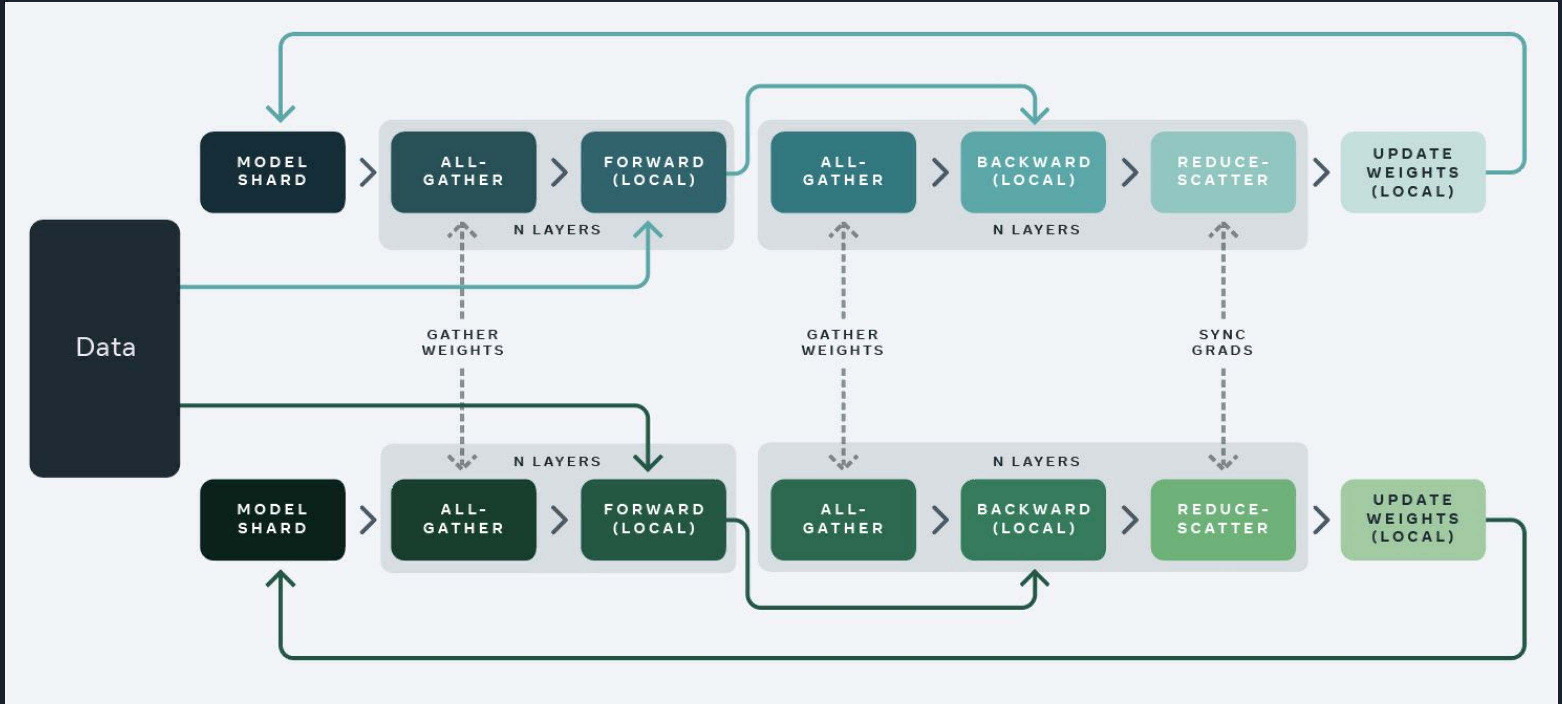


Overview of FSDP

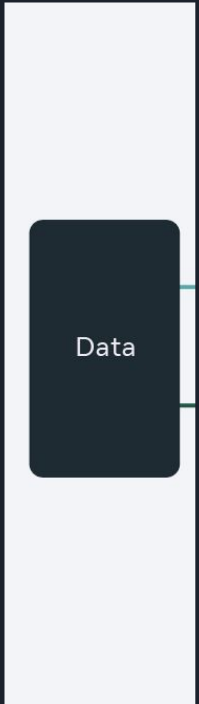
Fully Sharded Data Parallelism



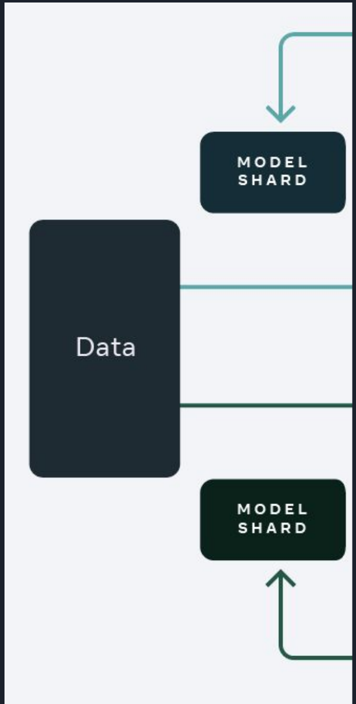
Overview of FSDP



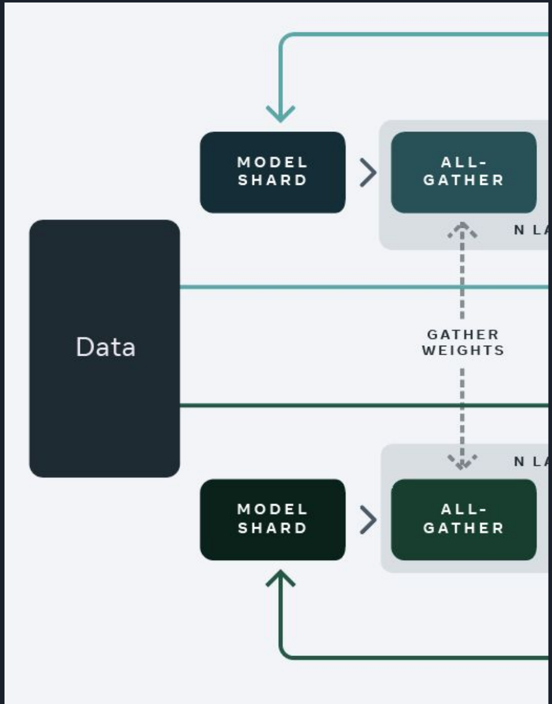
Overview of FSDP



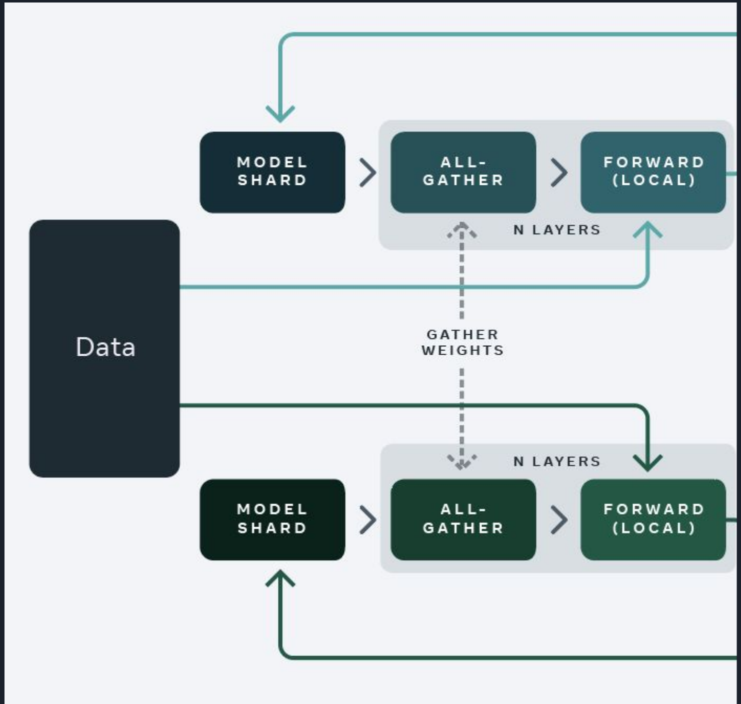
Overview of FSDP



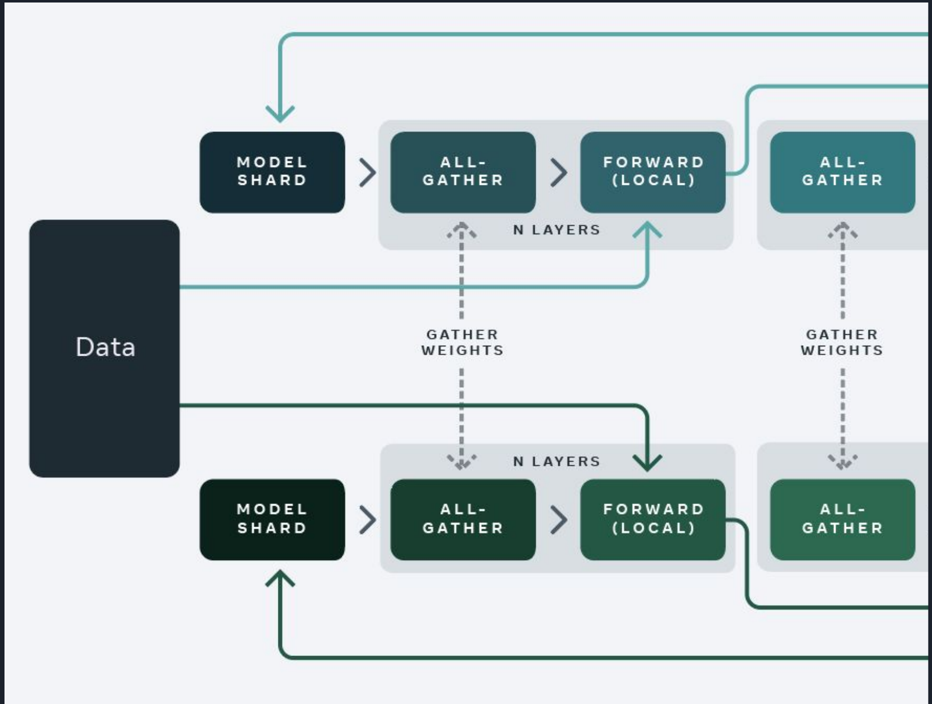
Overview of FSDP



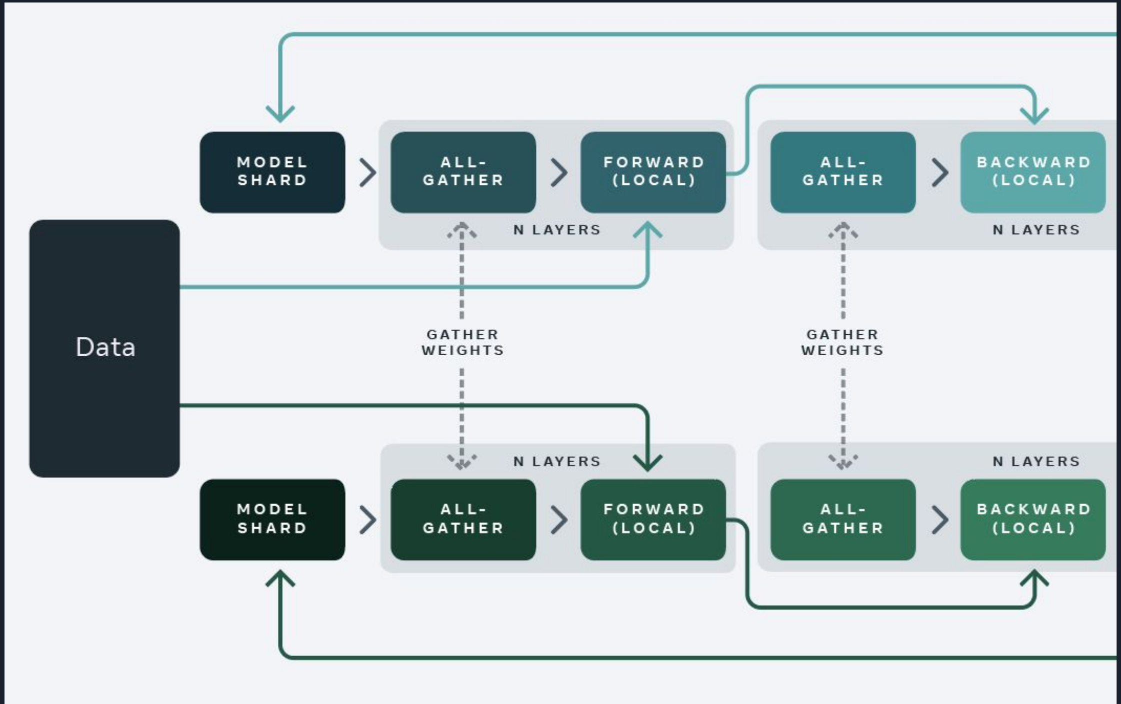
Overview of FSDP



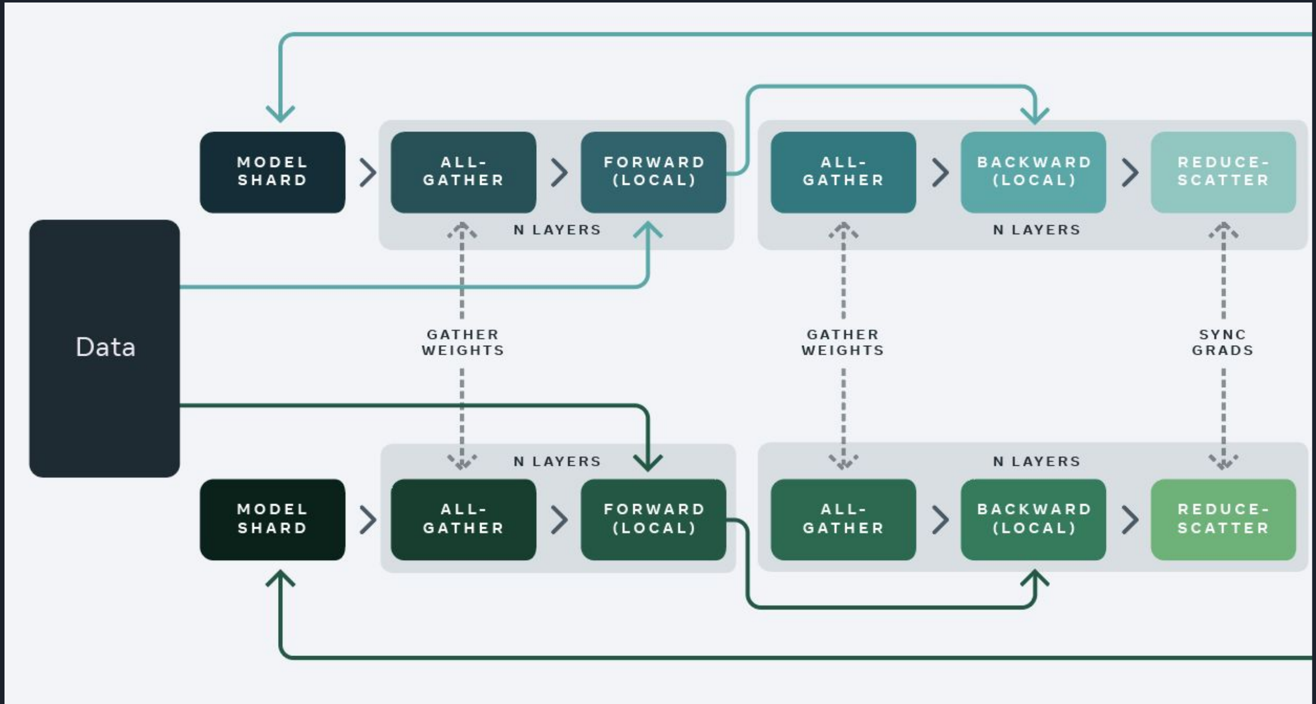
Overview of FSDP



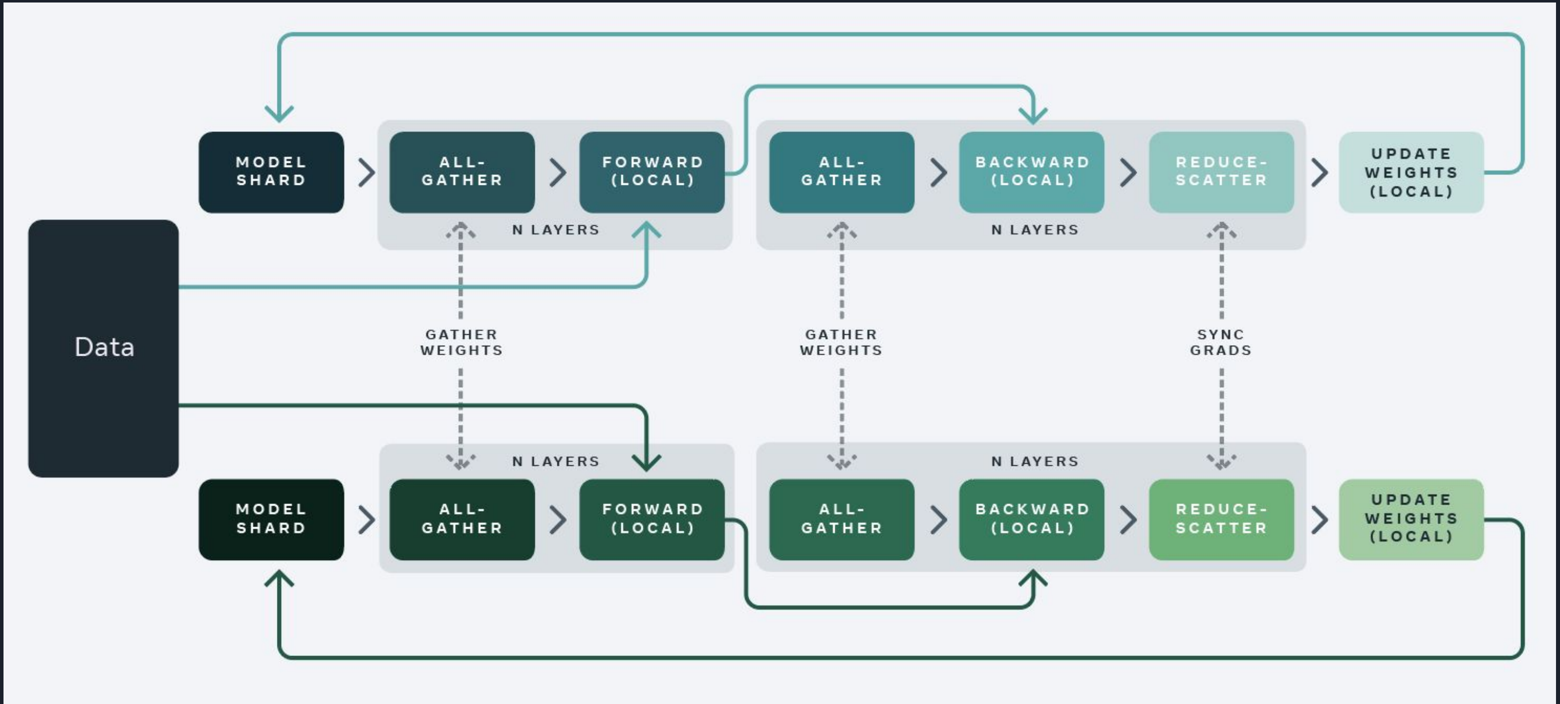
Overview of FSDP



Overview of FSDP



Overview of FSDP



Overview of FSDP

Fully Sharded Data Parallelism

Each GPU has a shard of the model (gradient and the optimizer state).

Split the dataset in batches and each gpu processes a different batch

Easy to use via *FullyShardedDataParallel* but less intuitive than *DistributedDataParallel*

Memory efficient but introduces more communication

FSDP in action

```
model=MLP(
  (fcs): Sequential(
    (0): Linear(in_features=20480, out_features=20480, bias=True)
    (1): ReLU()
    (2): Linear(in_features=20480, out_features=20480, bias=True)
    (3): ReLU()
    (4): Linear(in_features=20480, out_features=20480, bias=True)
    (5): ReLU()
    (6): Linear(in_features=20480, out_features=20480, bias=True)
    (7): ReLU()
    (8): Linear(in_features=20480, out_features=20480, bias=True)
    (9): ReLU()
    (10): Linear(in_features=20480, out_features=20480, bias=True)
    (11): ReLU()
    (12): Linear(in_features=20480, out_features=20480, bias=True)
    (13): ReLU()
    (14): Linear(in_features=20480, out_features=20480, bias=True)
  )
)
```

FSDP in action

```python
from torch.distributed.fsdp import FullyShardedDataParallel as FSDP

init_process_group(backend="nccl", rank=rank, world_size=world_size)

torch.cuda.set_device(rank)
device = torch.device(f"cuda:{rank}")
model = MLP()
model = model.to(device)
model = FSDP(model)
```

FSDP in action

```python
def train_step(
    data: torch.Tensor,
    target: torch.Tensor,
    model: nn.Module,
    optimizer: optim.Optimizer,
    criterion: nn.Module,
):
    optimizer.zero_grad()
    output = model(data)
    loss = criterion(output, target)
    loss.backward()
    optimizer.step()
```

FSDP in action

1. Constructor
   a. Shard model parameters and each rank only keeps its own shard

FSDP in action

1. Constructor
    a. Shard model parameters and each rank only keeps its own shard

2. Forward Call
    a. *all_gather* all shards from all ranks to recover the full parameter in current FSDP unit
    b. Run forward computation
    c. Discard parameter shards it has just collected

FSDP in action

1. Constructor
    a. Shard model parameters and each rank only keeps its own shard

2. Forward Call
    a. *all_gather* all shards from all ranks to recover the full parameter in current FSDP unit
    b. Run forward computation
    c. Discard parameter shards it has just collected

3. Backward call
    a. *all_gather* all shards from all ranks to recover the full parameter in current FSDP unit
    b. Run backward computation
    c. *Reduce_scatter* (sync) gradients
    d. Discard parameters

FSDP in action | What to shard

FULL_SHARD - Parameters, gradients, and optimizer states are sharded.

NO_SHARD - Nothing is shared - This is very similar for DDP

HYBRID_SHARD - Apply FULL_SHARD within a node, and replicate parameters across nodes.

FSDP in action | What to shard

```
model = FSDP(
    model,
    sharding_strategy=ShardingStrategy.FULL_SHARD,
)
```

FSDP in action | How to shard

Specify a policy for sharding layers

The policy can be based on the size (number of parameters) of the model or name of the model

This is very easy to get wrong

FSDP in action | How to shard

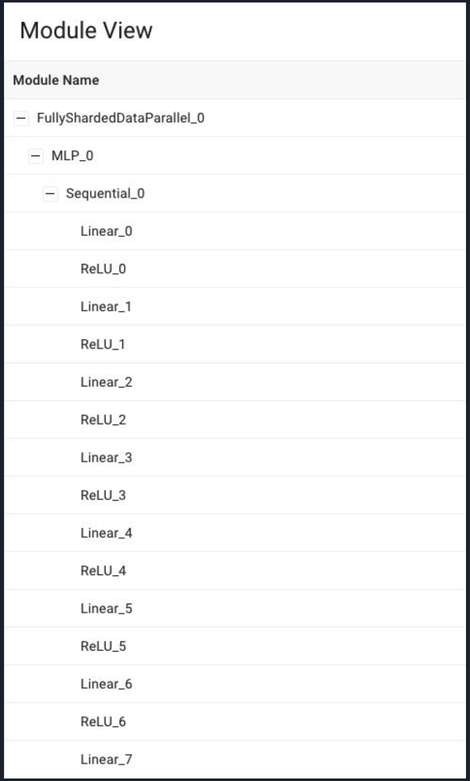
```
model = FSDP(
    model,
    sharding_strategy=ShardingStrategy.FULL_SHARD,
)
```

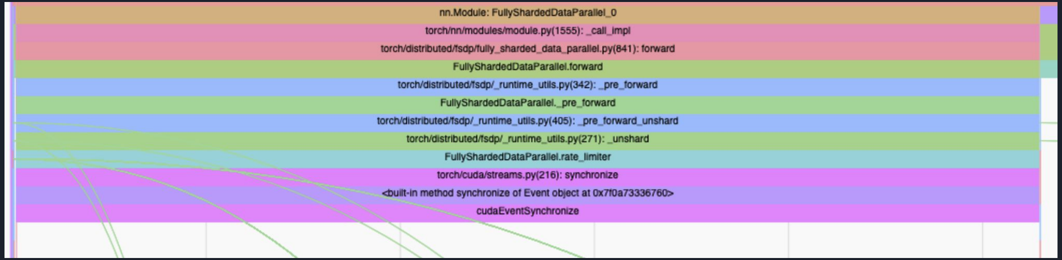
FSDP in action | How to shard

Module View

Module Name
FullyShardedDataParallel_0
MLP_0
Sequential_0
Linear_0
ReLU_0
Linear_1
ReLU_1
Linear_2
ReLU_2
Linear_3
ReLU_3
Linear_4
ReLU_4
Linear_5
ReLU_5
Linear_6
ReLU_6
Linear_7

FSDP in action | How to shard





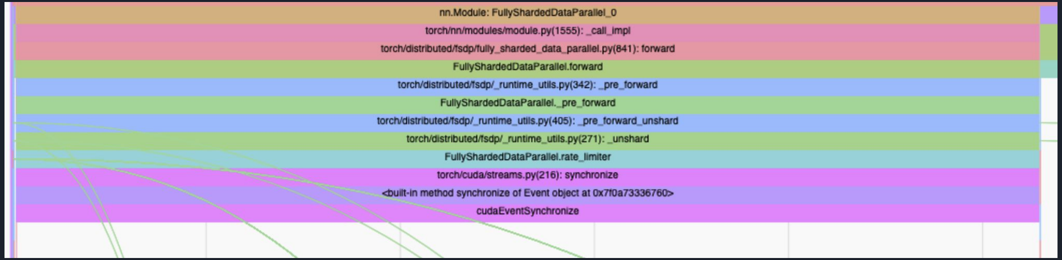
FSDP in action | How to shard

Module View

Module Name

- FullyShardedDataParallel_0
  - MLP_0
    - Sequential_0
      - Linear_0
      - ReLU_0
      - Linear_1
      - ReLU_1
      - Linear_2
      - ReLU_2
      - Linear_3
      - ReLU_3
      - Linear_4
      - ReLU_4
      - Linear_5
      - ReLU_5
      - Linear_6
      - ReLU_6
      - Linear_7

Peak Memory Usage: 43218.2MB



FSDP in action | How to shard

```
model = FSDP(
    model,
    sharding_strategy=ShardingStrategy.FULL_SHARD,
    auto_wrap_policy=ModuleWrapPolicy({nn.Linear}),
)
```

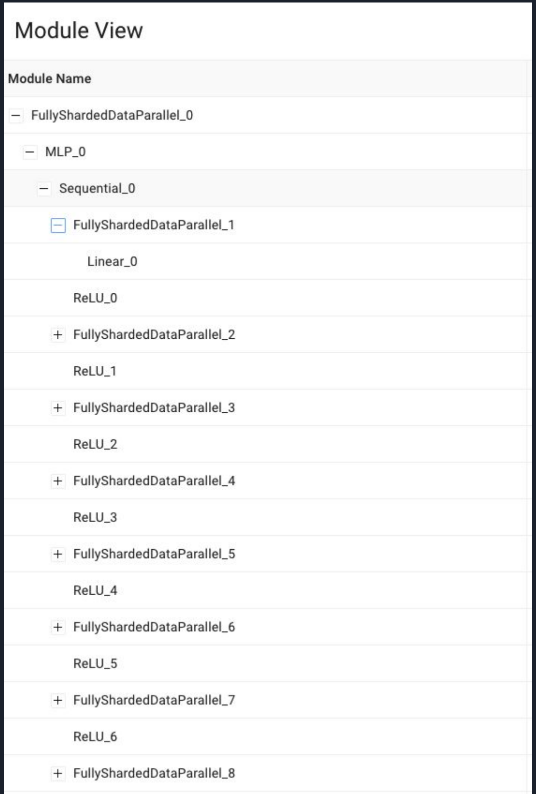
FSDP in action | How to shard

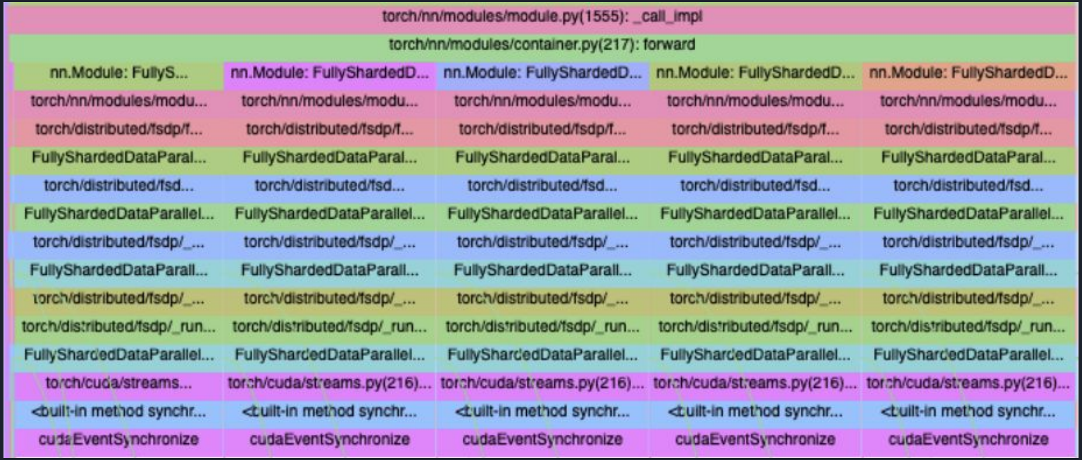
Module View

Module Name

- FullyShardedDataParallel_0
  - MLP_0
    - Sequential_0
      - FullyShardedDataParallel_1
        - Linear_0
      - ReLU_0
      - FullyShardedDataParallel_2
      - ReLU_1
      - FullyShardedDataParallel_3
      - ReLU_2
      - FullyShardedDataParallel_4
      - ReLU_3
      - FullyShardedDataParallel_5
      - ReLU_4
      - FullyShardedDataParallel_6
      - ReLU_5
      - FullyShardedDataParallel_7
      - ReLU_6
      - FullyShardedDataParallel_8

FSDP in action | How to shard





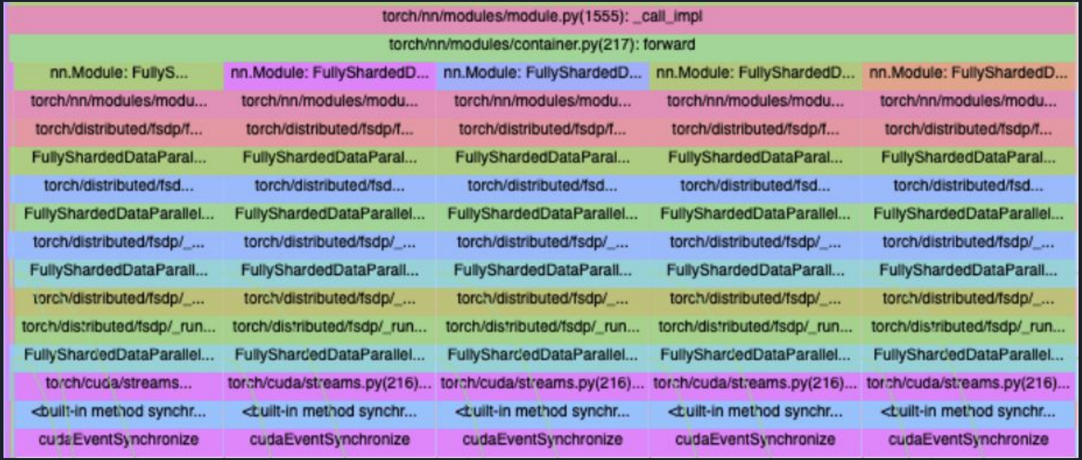
FSDP in action | How to shard

Module View

Module Name

- FullyShardedDataParallel_0
  - MLP_0
    - Sequential_0
      - FullyShardedDataParallel_1
        - Linear_0
      - ReLU_0
      - + FullyShardedDataParallel_2
      - ReLU_1
      - + FullyShardedDataParallel_3
      - ReLU_2
      - + FullyShardedDataParallel_4
      - ReLU_3
      - + FullyShardedDataParallel_5
      - ReLU_4
      - + FullyShardedDataParallel_6
      - ReLU_5
      - + FullyShardedDataParallel_7
      - ReLU_6
      - + FullyShardedDataParallel_8

Peak Memory Usage: 17616.6MB



FSDP in action | CPU Offload

```python
model = FSDP(
    model,
    sharding_strategy=ShardingStrategy.FULL_SHARD,
    auto_wrap_policy=ModuleWrapPolicy({nn.Linear}),
    cpu_offload=CPUOffload(offload_params=True),
)
```

FSDP in action | CPU Offload = False

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	52,133,082	100
Kernel	52,132,582	100
Memcpy	0	0
Memset	10	0
Runtime	0	0
DataLoader	0	0
CPU Exec	275	0
Other	216	0

FSDP in action | CPU Offload = False

Peak Memory Usage: 17616.6MB

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	52,133,082	100
Kernel	52,132,582	100
Memcpy	0	0
Memset	10	0
Runtime	0	0
DataLoader	0	0
CPU Exec	275	0
Other	216	0

FSDP in action | CPU Offload = True

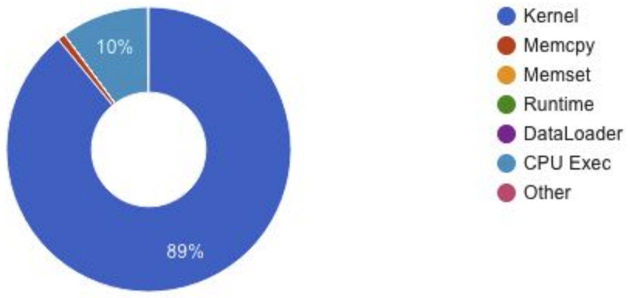
Peak Memory Usage: 16016.6MB

FSDP in action | CPU Offload = True

Peak Memory Usage: 16016.6MB

Execution Summary

Category	Time Duration (us)	Percentage (%)
Average Step Time	62,136,475	100
Kernel	55,323,892	89.04
Memcpy	531,020	0.85
Memset	11	0
Runtime	0	0
DataLoader	0	0
CPU Exec	6,197,215	9.97
Other	84,338	0.14



FSDP in action | Other Options

1. forward_prefetch

2. limit_all_gathers / rate limiter

3. mixed_precision

FSDP in action | Profiling

1. Standard Pytorch profiling techniques apply [5, 6]
2. Look out for
   a. time spent in sharding or unsharding parameters during forward and backward passes.
   b. How often and how long all-gather operations take to complete, especially across multiple nodes.

FSDP in action | Common Pitfalls

1. Ensure consistent initialization using say *sync_module_states*

2. Use *backward_prefetch*, *forward_prefetch* and *limit_all_gathers* to reduce network latency

3. Use *cpu_offload*, *mixed_precision* and activation checkpointing to reduce memory usage

4. Uneven GPU Loads due to uneven sharding of model

5. Checkpointing the models

Next Step

1. Getting Started with FSDP — PyTorch Tutorials
2. Advanced Model Training with FSDP — PyTorch Tutorials
3. PyTorch FSDP Tutorials - YouTube

Beyond FSDP

1. FSDP2
2. Pipeline Parallel
3. Context Parallel
4. Tensor Parallel

References

1. Strong Scaling Hypothesis
2. Scaling Laws for Neural Language Models
3. DeepSpeed
4. FSDP vs DeepSpeed
5. PyTorch Profiler — PyTorch Tutorials
6. torch.profiler — PyTorch 2.5 documentation

Thank you!

https://shagunsodhani.com/talks/

ODSC 2024

@shagunsodhani