



CONFERENCE

—NORTH AMERICA 2026—



FlexShard: Pluggable Sharding Placements for Matrix Optimizers and Block-wise Quantization

Wei Feng, Tianyu Liu, Anshul Sinha, Shuhua Yu, Ailing Zhang, Edward Yang
Aditya Venkataraman, Davide Italiano

2026 Oct

What's FlexShard



- Data Parallelism API for sharding model and optimizer state
 - API to manage transitions between fwd/bwd compute sharding and storage sharding
 - E.g., TP Shard(0|1) at fwd/bwd compute sharding, Owned at storage sharding
- Bucket-centric API: layer-by-layer param unshard and gradient reduction
- Offers extensible sharding strategies for params and grads
 - Megatron DDP, FSDP1: flat buffer
 - FSDP2 style: per-param shard(0|1)
 - Owned and BlockShard for Muon and blockwise quantization
 - User-defined placements
- Optimized runtime for overlapped communication and computation
- Intended as replacements for FSDP2/1 and SimpleFSDP
 - bitwise identical to FSDP2 on Shard(0), FSDP1 on FlatShard

FlexShard Motivation 1: Muon Optimizer Requires Full Parameter

Dense Params

FSDP2

Shard(0) — scattered over every rank



FlexShard

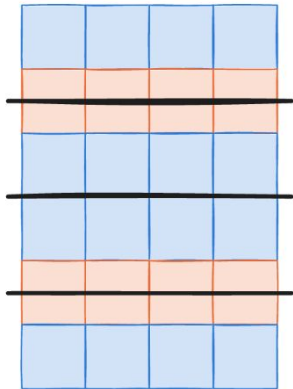
Owned — one rank holds it all



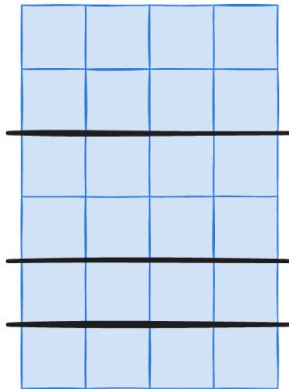
- FSDP2/Megatron-DDP-style: communication required for Muon
- FlexShard: new **Owned/BlockShard** placements
 - Enable Communication-free Muon!

FlexShard Motivation 2: Align Sharding with Block-wise Quantization

Shard(0) — equal shards



BlockShard — whole tiles



- FSDP2-style sharding precludes/complicates blockwise quantized AG
- FlexShard: new **BlockShard** placement to align sharding boundaries
 - Enable quantized AGs, reducing communication volume

FlexShard Motivation 3: User-defined placements

- Allow for new param/grad placements
 - **without** re-inventing the communication runtime
- Extensible Placement APIs for shard/collective/unshard contracts
- FlexShard supports many in-built strategies with a single communication runtime
 - Megatron-DDP, FSDP1, veScale-FSDP: flattened buffer + view params/grads
 - FSDP2, DeepSpeed Zero-3: Shard(0)/Shard(1) + copy-in/-out params/grads
 - Owned (**new**)
 - BlockShard (**new**)

FlexShard for nn.Module: bucket-centric API

bucket: which params/grads should be grouped together for a single all-gather / reduce-scatter

storage sharding: how params/grads are sharded outside of fwd/bwd

fwd/bwd compute sharding: Replicate for DP-only, and Shard(0) | Shard(1) for TP

```
flex_shard(model, buckets=buckets)
buckets = [
    # dense params: attention, shared experts, DP mesh
    BucketSpec([f"layers.{i}.*attention.*", ...],
                placement_fn=Owned | BlockShard | Shard(0) | Shard(1),
                mesh=dp_mesh),
    # sparse params: routed experts, EDP mesh
    BucketSpec([f"layers.{i}.moe.routed_experts.inner_experts.*"],
                placement_fn=Shard(0) | Shard(1),
                mesh=edp_mesh),
]
```

FlexShard for Optimizer: DistMuon

storage sharding: how params/grads are sharded in-between fwd/bwd and optimizer step

optimizer compute sharding: how params/grads are sharded during optimizer step

```
optimizer = build_dist_muon(
    params,
    compute_sharding_by_fqn=compute_sharding_by_fqn,
    bucket_configs=bucket_configs,
)

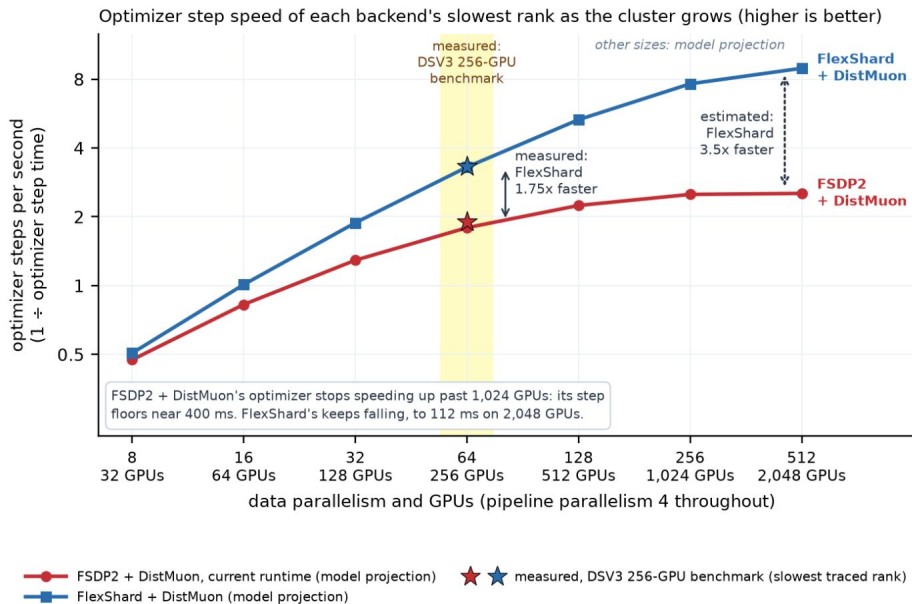
compute_sharding_by_fqn = expand_fqn_patterns(model, {
    # dense params: attention, shared experts
    "layers.*.attention.wq": ComputeLayout({"dp_shard": BlockShard(0, q_head_dim)}),
    "layers.*.attention.wkv": ComputeLayout({"dp_shard": BlockShard(0, kv_rows_per_head)}),
    "layers.*.attention.wo": ComputeLayout({"dp_shard": Owned()}),

    # routed experts: independent matrices along the expert dimension.
    "layers.*.moe.routed_experts.inner_experts.w*": ComputeLayout(
        {"edp": Shard(0), "ep": Shard(0)},
        shard_order_by_tensor_dim={0: ("ep", "edp")},
    ),
})
```

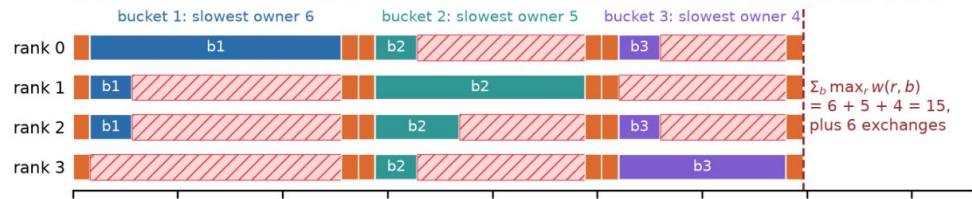
Performance: FlexShard + DistMuon

- **Baseline:** FSDP2 **Shard(0)** model + DTensor Muon with per-param all-gather
- **Communication-efficient Muon: x1.74 over baseline**
 - FSDP2 **Shard(0)** model + FlexShard Muon with bucketed A2A
 - Perfs gain comes from comm bucketing and overlapping with compute
- **Communication-free Muon: x1.75 over communication-efficient Muon**
 - FlexShard **Owned | BlockShard** model + FlexShard Muon with no comms
 - Perfs gains comes from eliminating communication during muon compute

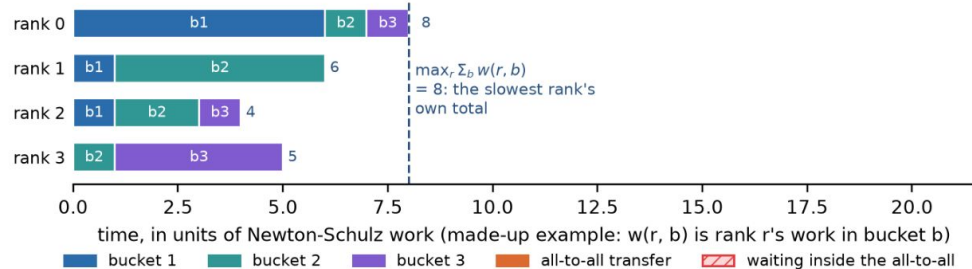
Performance: FlexShard + DistMuon



FSDP2 + DistMuon: every bucket's exchange waits for that bucket's slowest owner

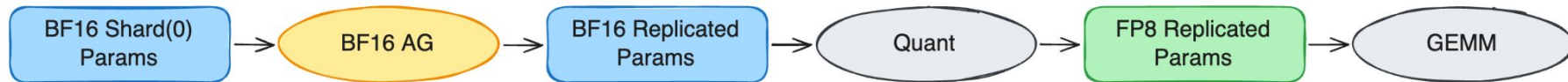


FlexShard + DistMuon: each rank runs only the matrices it owns; nothing to wait for

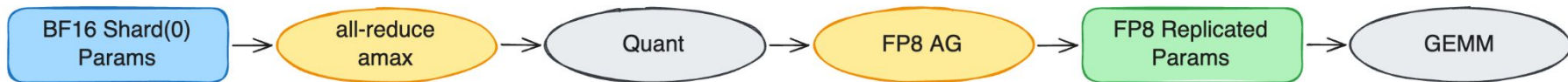


Performance: FlexShard + Blockwise Quantization

Baseline: BF16 all-gather and post-all-gather quantization



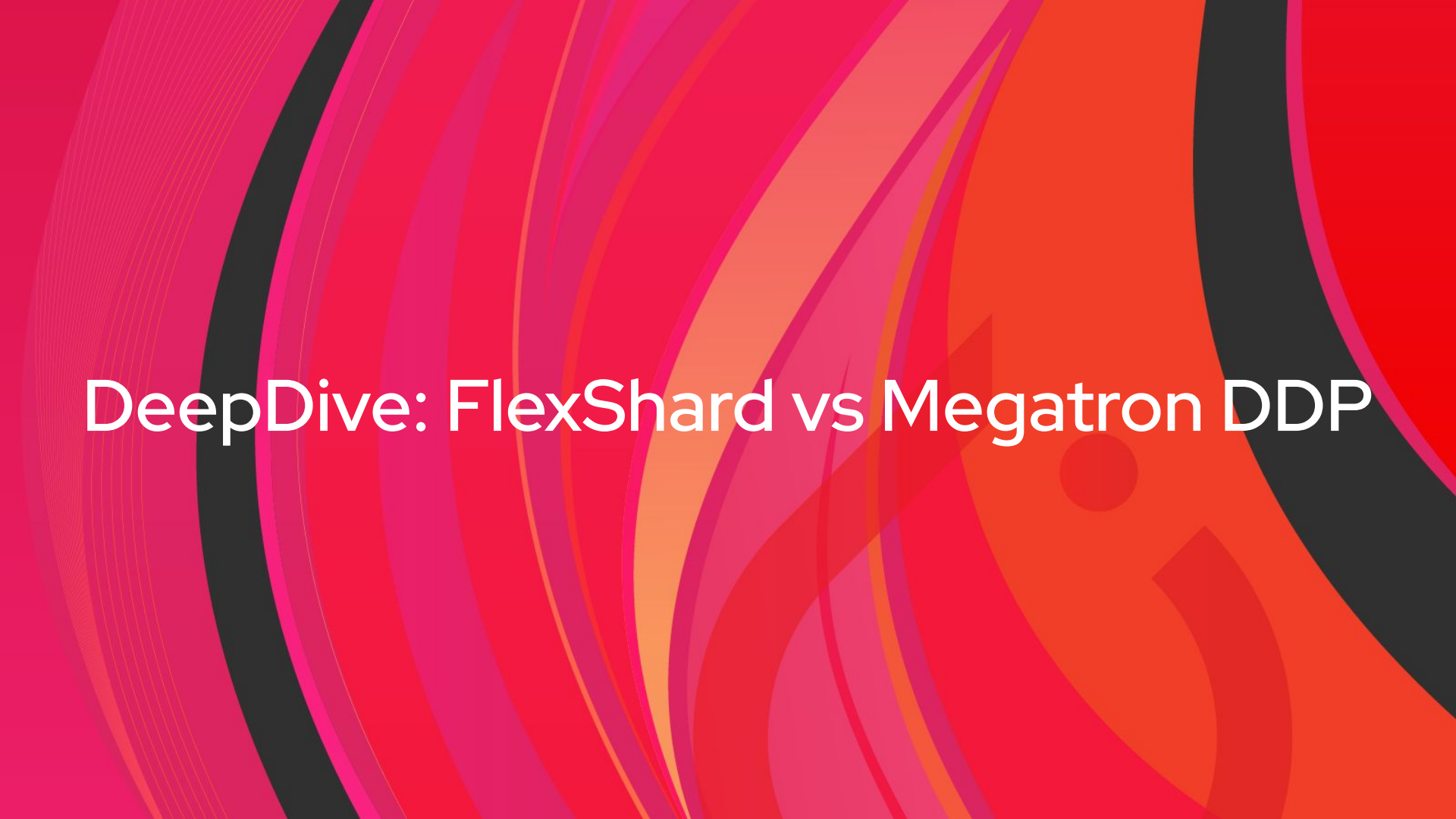
FSDP2: all-reduce amax, FP8 all-gather



FlexShard: comm-free amax, FP8 all-gather: **+20% TPS on comm-bound, +10% on compute-bound**



DeepDive: FlexShard vs Megatron DDP



FlexShard

vs

Megatron DDP

Replicate

Shard(0)

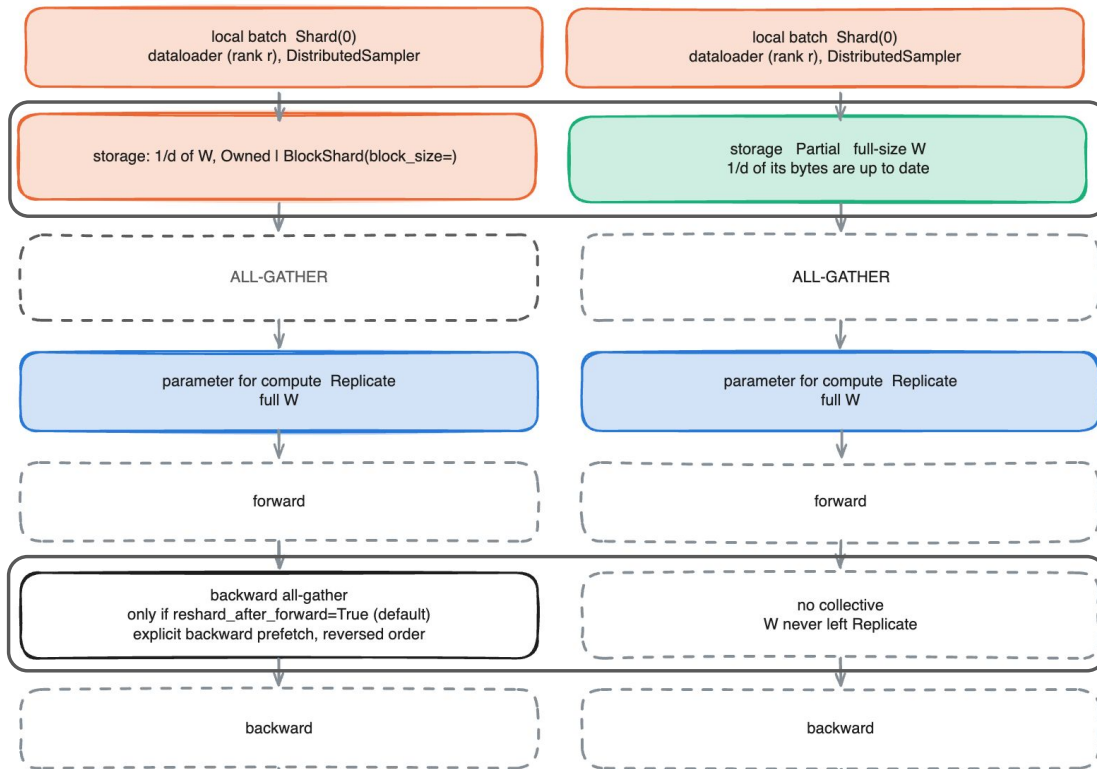
Partial

Fwd

FlexShard shards params at the boundary of head/experts/quantization blocks

AG/RS bucket by fqns, not buffer size

PP turns off reshard_after_forward between micro-batches



FlexShard

vs

Megatron DDP

Replicate

Shard(0)

Partial

Bwd

backward

backward

gradient Partial
sum over local batch

gradient Partial
sum over local batch

reduce-scatter
Partial -> Owned | BlockShard

reduce-scatter
Partial -> Shard(0)

gradient Owned | BlockShard
1/d kept, full buffer freed

gradient Shard(0)
1/d used, full buffer kept

optimizer state Owned | BlockShard
1/d states

optimizer state Shard(0)
1/d states

**FlexShard removes
communication in
Optimizer**

Shard(0)-to-full param comms during optimizer step

resident BUFFER bytes per rank
model state 1/d · optim state 1/d

resident BUFFER bytes per rank
model state FULL · optim state 1/d

difference in optimizer states

- sharding at the boundary of heads and experts

difference in model state

- sharded vs replicated

Recap



- FlexShard:
 - Co-design sharding between model and optimizer: Owned and BlockShard for Muon
 - Align model sharding with blockwise quantization: BlockShard for (mx)fp8
 - Custom placements describe tensor layout programmatically: unshard params, reduce grads
 - One flex_shard API across eager and compile
- Availability in OSS
 - FlexShard for optimizer: communication-efficient Muon: landed in [TorchTitan](#)
 - FlexShard for model: communication-free Muon and blockwise quantization [meta-pytorch/flex_shard](#)
 - FlexShard on Megatron-LM (our fork): FlexShard+DistMuon replacing DDP+DistributedOptimizer [Megatron-FlexShard](#)