MLSys 2026

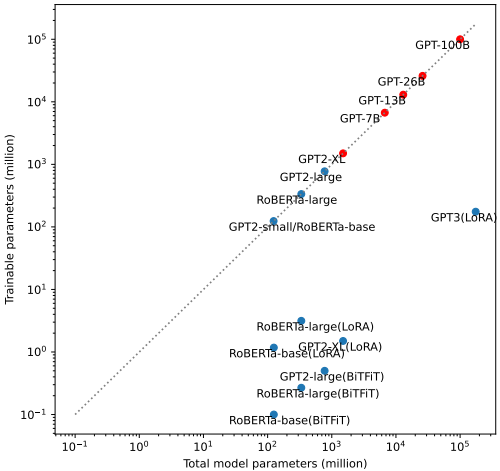
Zero Redundancy Distributed Learning with Differential Privacy

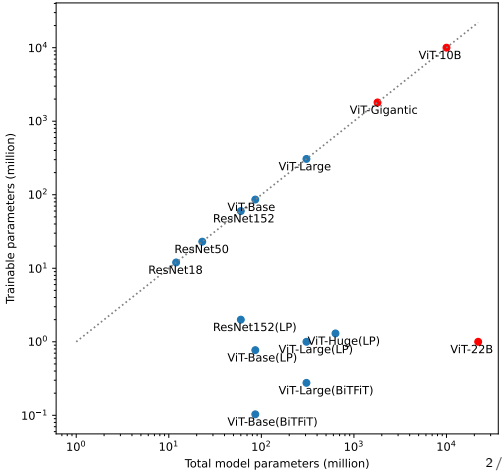
Making DP training scale like standard ZeRO

Zhiqi Bu Justin Chiu Ruixuan Liu Sheng Zha George Karypis

Motivation: DP needs large-model distributed training

- Single GPU DP can be efficient with ghost clipping / book-keeping.
- Multi-GPU DP must handle parameter/gradient sharding.
- Target scale: 1B to 100B trainable parameters.





Main contributions

1. **DP-ZeRO**: DP training with ZeRO-style memory, computation, and communication efficiency.
2. **Local DP back-propagation**: per-sample clipping/noising without disrupting ZeRO's communication schedule.
3. **DP mixed precision**: bf16/fp16 support, with the key rule that DP should not use standard loss scaling.
4. **Scale**: full-parameter DP training beyond 1B parameters, up to GPT-100B and 256 GPUs.
5. **Implementation**: base-layer back-propagation rewrites instead of fragile hook composition.

Differential privacy objective: privatize gradients

For group m, DP optimization replaces the standard gradient by

$$G^{[m]} = \sum_i c^{[m],i} g^{[m],i} + \sigma_{\mathrm{DP}} \mathcal{N}(0, I).$$

Two operations

- **Per-sample clipping**: bound each sample's influence.
- **Gaussian noising**: obtain (ϵ, δ)-DP through privacy accounting.

Design pressure in distributed training

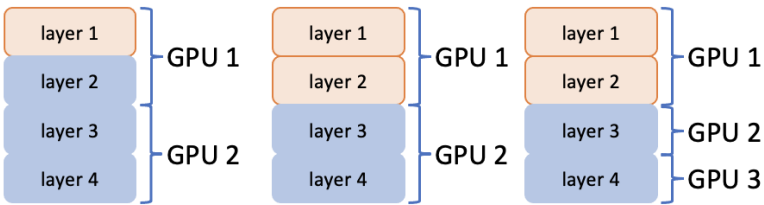
The clipping factor $c^{[m],i}$ depends on per-sample gradient norms. If the required norm spans shards on different GPUs, DP introduces extra communication.

ZeRO removes redundant model states

Stage	Sharded state	Per-GPU memory intuition
ZeRO-1	optimizer states	$(4 + 12/N_d)\Psi_{\text{model}}$
ZeRO-2	+ gradients	$(2 + 14/N_d)\Psi_{\text{model}}$
ZeRO-3	+ parameters	$(16/N_d)\Psi_{\text{model}}$

Why it matters for DP

DP modifies the gradient path. ZeRO modifies parameter, gradient, and optimizer-state placement. DP-ZeRO must respect both.



The core incompatibility

Mathematical partition

DP defines groups for clipping: all-layer, layer-wise, or finer. This changes the clipping factor and therefore the update.

Hardware partition

ZeRO shards tensors across GPUs for memory. This should not change the update, only where tensors live.

Rule of thumb

Make each mathematical clipping group local to one hardware shard whenever possible, so clipping does not require cross-GPU norm aggregation.

Locality of DP back-propagation

Standard ZeRO iteration, layer by layer:

$$(\texttt{all-gather} \to \texttt{forward} \to \texttt{all-gather} \to \texttt{backward} \to \texttt{reduce}) \times L.$$

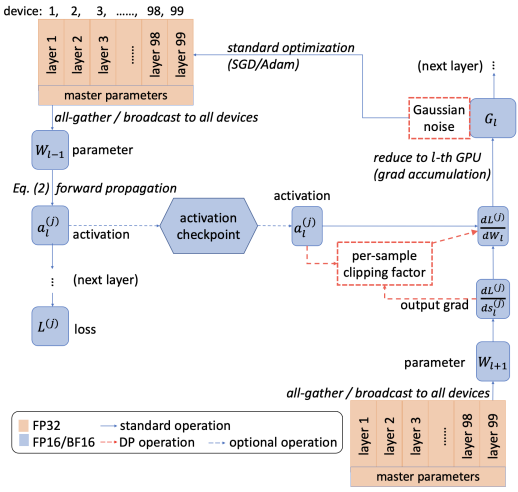
DP-ZeRO invariant

DP operations are inserted *inside the local backward computation*:

$$\texttt{backward} = \texttt{output grad} \to \texttt{clip factor} \to \texttt{parameter grad} \to \texttt{noise}.$$

This keeps ZeRO's global communication pattern unchanged.

Overall algorithm



Layer-level math: linear layer example

Forward pass is unchanged:

$$s_l = a_l W_l + b_l, \qquad a_{l+1} = \phi_l(s_l).$$

Output gradient:

$$\frac{\partial L}{\partial s_l} = \frac{\partial L}{\partial s_{l+1}} W_{l+1} \circ \phi_l'(s_l).$$

DP parameter gradient:

$$\frac{\partial \sum_i c_i L_i}{\partial W_l} + \sigma \mathcal{N}(0, I) = a_l^\top \operatorname{diag}(c_1, \ldots, c_B) \frac{\partial L}{\partial s_l} + \sigma \mathcal{N}(0, I).$$

Implementation point

Compute c_i locally from activation and output-gradient statistics; add noise before ZeRO's gradient reduction.

Why DP-ZeRO approaches ZeRO speed at scale

Key ratio

$$\frac{\text{DP-ZeRO speed}}{\text{ZeRO speed}} \approx \frac{\text{standard backprop} + \text{forward} + \text{communication}}{\text{DP backprop} + \text{forward} + \text{communication}}.$$

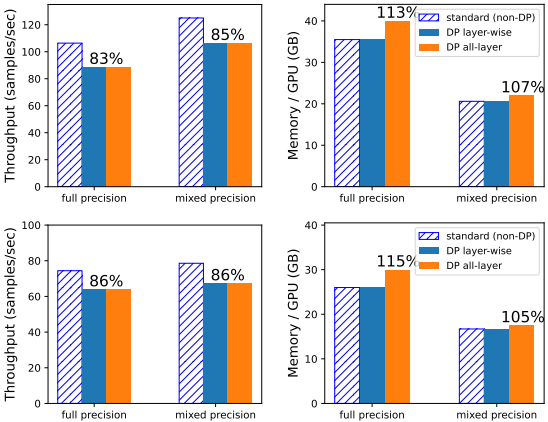
- Multi-node training increases the communication share of runtime.
- ZeRO-3 all-gathers sharded parameters, increasing non-DP overhead that DP does not change.
- Activation checkpointing increases forward recomputation, again shrinking the relative DP overhead.
- PEFT reduces Ψ_{train}, so DP clipping and parameter-gradient work become much smaller.

Counterintuitive consequence

The more communication- or forward-dominated the baseline becomes, the closer DP-ZeRO is to standard ZeRO in relative speed.

Memory efficiency: avoid per-sample gradient materialization

- Use mixed ghost norm / book-keeping instead of instantiating all per-sample gradients.
- Prefer layer-wise clipping to avoid storing all output gradients.
- With many GPUs and gradient accumulation, per-GPU micro-batch B can be small; when $B = 1$, per-sample gradients are effectively free.



Mixed precision: DP should not use standard loss scaling

Standard fp16

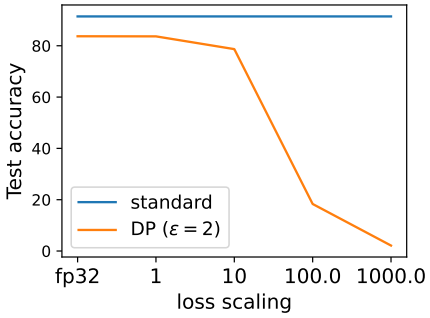
Scale loss up before backprop, scale gradients down before optimizer step, preventing underflow.

DP fp16/fp32 issue

Per-sample clipping already rescales gradients. Standard loss scaling can cause overflow in norm computation or over-shrink the privatized gradient.

Recommendation

Use bf16 mixed precision without loss scaling when possible.



Hook-based DP is fragile with ZeRO

DP libraries	ZeRO libraries
Often use module hooks: forward hooks and backward hooks.	Often use tensor hooks to trigger communication and state movement.

- Skipping non-private gradient computation may skip ZeRO tensor hooks.
- Hook counts can become large: a 100-layer network with weight+bias can need 100 module hooks + 200 tensor hooks.
- Per-sample gradients and book-kept tensors may be sharded by ZeRO-2/3, forcing extra communication.

Preferred implementation: rewrite base-layer backprop

```
class DPLinearFunction(Function):
    def forward(ctx, input, weight, bias=None):
        ctx.save_for_backward(input, weight, bias)
        return F.linear(input, weight, bias)

    @staticmethod
    def backward(ctx, grad_output):
        input, weight, bias = ctx.saved_tensors
        grad_input = grad_output.mm(weight)
        # compute per-sample norm from input and grad_output
        # compute clip_factor
        # aggregate clipped parameter gradient
        # add Gaussian noise before ZeRO reduction
        return grad_input, grad_weight, grad_bias
```

Rewrite base layers once: Linear, Embedding, Conv, LayerNorm, GroupNorm, etc. Advanced modules such as attention, LoRA, MoE routers, and SwiGLU inherit DP support through base layers.

User interface and optimizer compatibility

```
model = build_model(...)
privacy_engine = PrivacyEngine(
    model,
    batch_size=256,
    sample_size=50000,
    epochs=3,
    target_epsilon=3,
)
# optimizer is untouched: AdamW, SGD, ZeRO optimizer, etc.
```

Design choice

The codebase does not modify the optimizer. DP is implemented in model-layer backward passes, so arbitrary optimizers and ZeRO implementations can be supported.

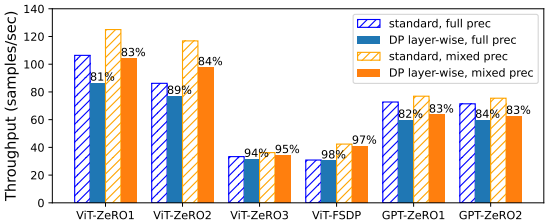
Experimental setup

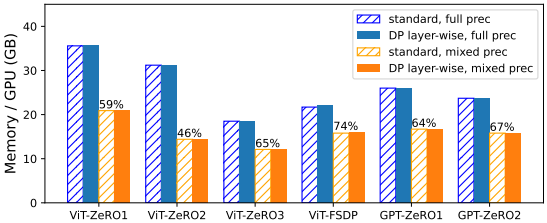
- Architectures: ResNet, ViT, GPT.
- ZeRO implementations: DeepSpeed, FSDP, MiCS-style settings.
- Settings: full fine-tuning and PEFT; fp32 and mixed precision; layer-wise and all-layer clipping.
- Scale: single node to 256 GPUs; 7B to 100B trainable parameters.
- Default: AdamW, mixed precision, layer-wise clipping, micro-batch $B = 4$, A100 GPUs.

Evaluation focus

System efficiency: throughput, per-GPU memory, scalability, and compatibility. Accuracy checks validate that DP-ZeRO preserves the intended optimization behavior.

Results: ZeRO stages close the DP overhead gap

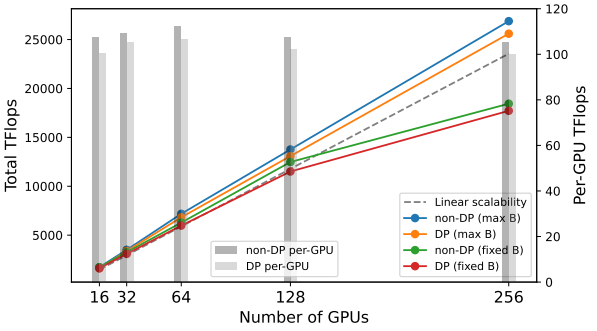


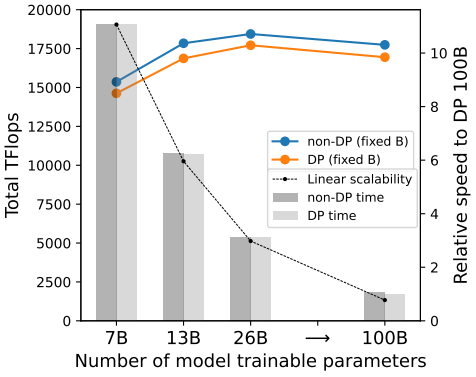


Takeaway

As ZeRO stage increases, communication and sharding overhead dominate more of the baseline, so DP-ZeRO throughput approaches standard ZeRO.

Results: Large-scale scalability





DP-ZeRO3 scales to 100B parameters and up to 256 GPUs.

Summary and takeaways

1. DP-ZeRO makes DP a local backward-pass transformation rather than a global communication rewrite.
2. The central correctness condition is alignment between DP mathematical groups and hardware shards.
3. Rewriting base-layer backprop avoids hook conflicts and supports advanced architectures.
4. Mixed precision is feasible, but DP changes the loss-scaling story: bf16 without loss scaling is the clean path.
5. Empirically, DP-ZeRO approaches standard ZeRO efficiency and enables DP training at 100B-parameter scale.

`https://github.com/awslabs/fast-differential-privacy`