



CONFERENCE

—NORTH AMERICA 2026—

**DeepSpeed Is Not Just ZeRO:
Tensor, Sequence, and Expert Parallelism with
ZeRO-Level Usability**

Masahiro Tanaka
DeepSpeed



TL;DR

DeepSpeed now matches or outperforms other SoTA frameworks on LLM training performance

Zero-Lever Usability

Masahiro Tanaka
DeepSpeed

About DeepSpeed

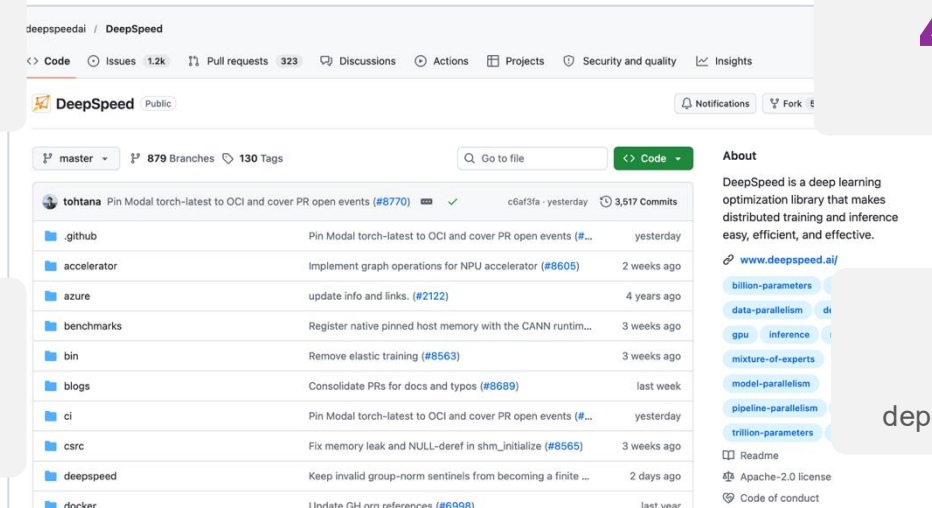
Pioneer library for distributed training

43k+

GitHub stars

1M+

PyPI downloads / month



400+

contributors

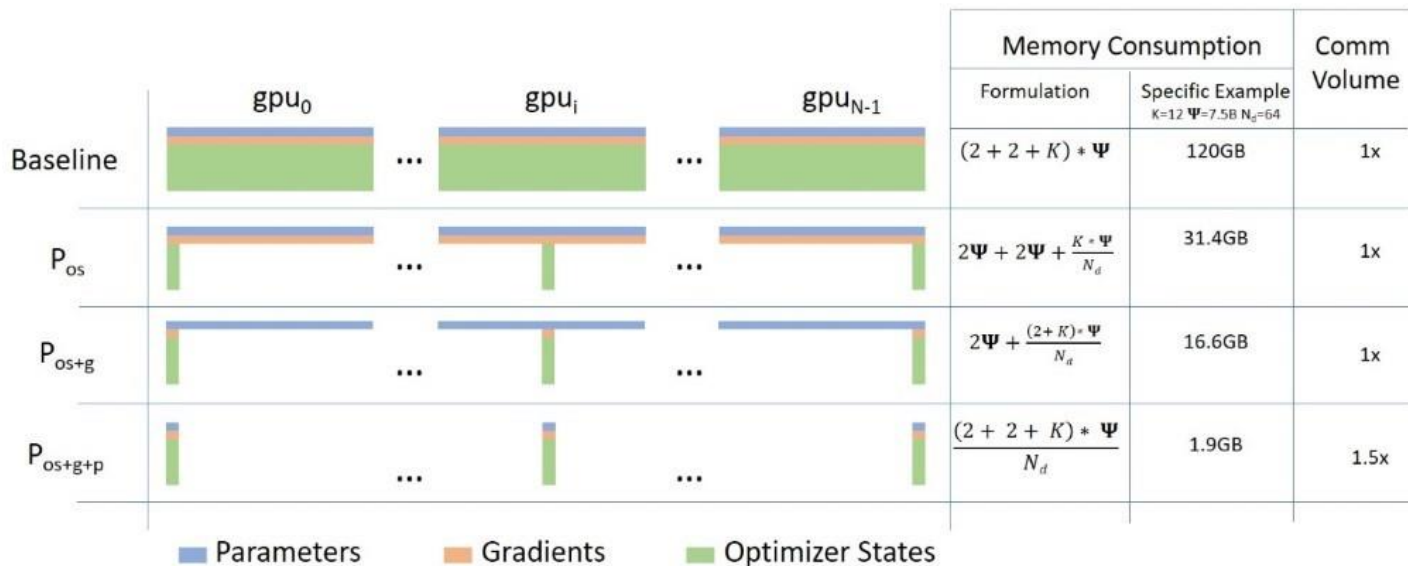
4000

dependent repositories

github.com/deepspeedai/DeepSpeed

DeepSpeed ZeRO

- **Partitioning** DL state across data parallel GPUs
- Now in FSDP, Megatron, TorchTitan, ...



Minimum Code Change

Write a JSON config

```
{
  "train_micro_batch_size_per_gpu": 4,
  "gradient_accumulation_steps": 8,
  "bf16": { "enabled": true },
  "optimizer": {
    "type": "AdamW",
    "params": { "lr": 2e-5 }
  },
  "zero_optimization": {
    "stage": 3,
    "offload_optimizer": { "device": "cpu" }
  }
}
```

ds_config.json

Init and train

```
import deepspeed
from transformers import AutoModelForCausalLM

model = AutoModelForCausalLM.from_pretrained(name)

model, optimizer, _, _ = deepspeed.initialize(
    model=model,
    model_parameters=model.parameters(),
    config="ds_config.json",
)

for batch in data_loader:
    loss = model(**batch).loss
    loss.backward()
    model.step()
```

Only change

DeepSpeed Integration

- Integrated with many other frameworks:



Transformers



Accelerate



Lightning



Axolotl



OPENRLHF

With HF Trainer

```
from transformers import Trainer, TrainingArguments
args = TrainingArguments(
    output_dir="out",
    bf16=True,
    deepspeed="ds_config.json",
)
trainer = Trainer(model=model, args=args,
                  train_dataset=dataset)
trainer.train()
```

Only change

DeepSpeed = ZeRO?

What we hear

“ZeRO? FSDP has that now.”

“DeepSpeed is only for HF integration.”

“For large models, you use Megatron with N-D parallelism.”

What we chose

Fair - We chose generality

- ZeRO: any model, simple config.
- TP / EP / SP: model-specific

The label stuck: DeepSpeed = ZeRO.

DeepSpeed Is Not Just ZeRO — Anymore

Added the parallelisms for large models

EP

AutoEP

TP

AutoTP

SP

AutoSP

PP

Ray-based

State-of-the-art performance

EP

Faster than Megatron / NeMo on MoE training

Qwen3-30B-A3B, MiniMax-M3-428B-A23B, GLM-5.2-744B-A40B

TP

Matches Megatron TP

Qwen3-32B (dense)

SP

Up to 3× longer context than Ulysses / Ring

Llama 3.1 8B, 8× A100

PP

Faster than Megatron-Bridge and TorchTitan on VLM training, on par with Megatron-MIMO

Qwen3.5-27B, 35B-A3B

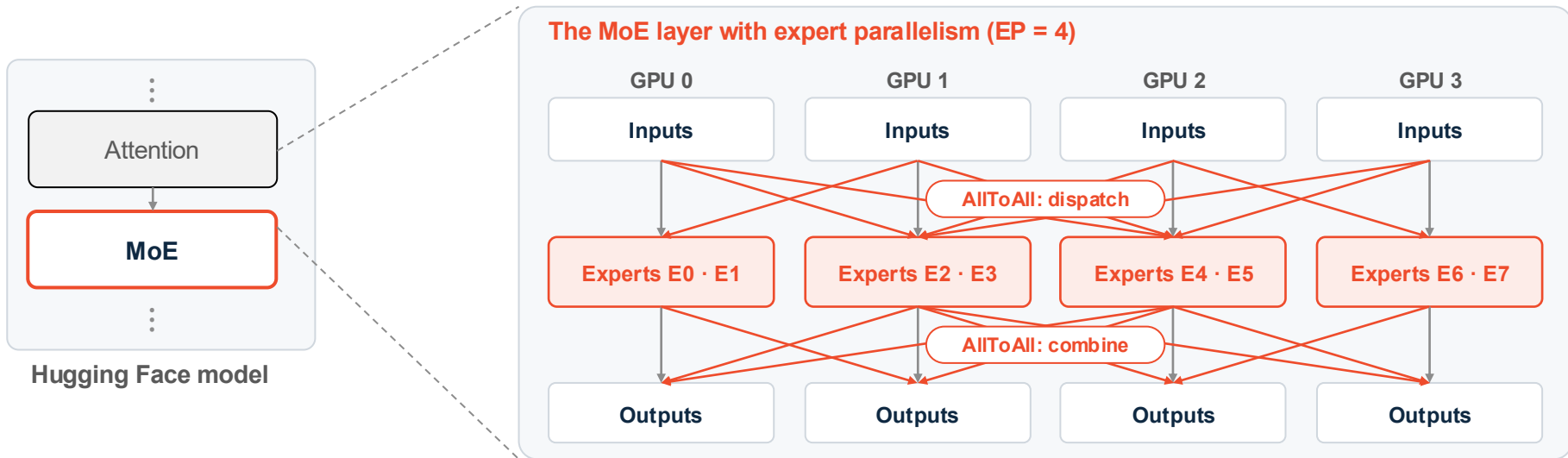
```
{
  "bf16": { "enabled": true },
  "zero_optimization": {
    "stage": 1
  },
  "expert_parallel": {
    "autoep_size": 16
  }
}
```

Same for TP and SP

AutoEP (Expert Parallel)

with **Peng Du, Han Yu**, and many others!

- **Expert parallelism (EP)**: spreads experts of an MoE layer across GPUs
- **Detect and replace**: pattern-matches the MoE blocks of a Hugging Face model and swaps in its own EP layer in place



Benchmark: Single-Node MoE

Model: Qwen3-30B-A3B
HW: 8× H100 80GB
Seq len: 4,096
Parallel: EP8/TP1/PP1
Batch: μ4 · GA16 · GBS512

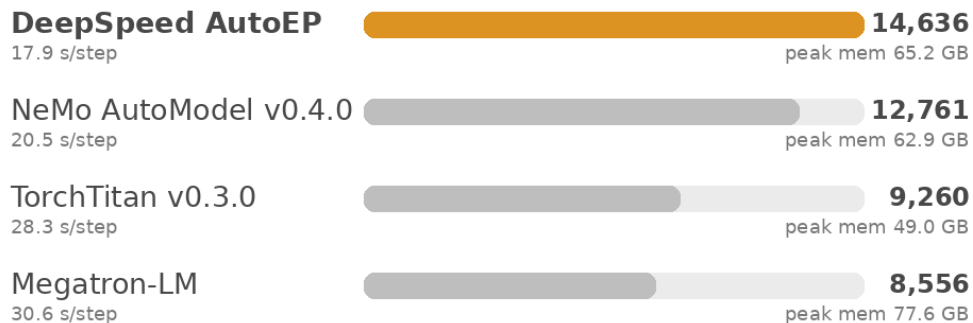
1.15×
vs. AutoModel

1.58×
vs. TorchTitan

throughput / GPU

Throughput per GPU

(higher is better)



Note: Megatron is memory-constrained here (77.6 vs. 65.2 GB/GPU), which may depress its measured performance.

Benchmark: Multi-Mode Large-Model MoE

HW: H200 · 8 GPUs/node
Config: best per framework
Seq len: 65,536
Weights: real checkpoints

Up to 1.40× speed-up vs. Megatron

- 2 / 3 benchmarks faster
- Scales to 744B-class MoE training

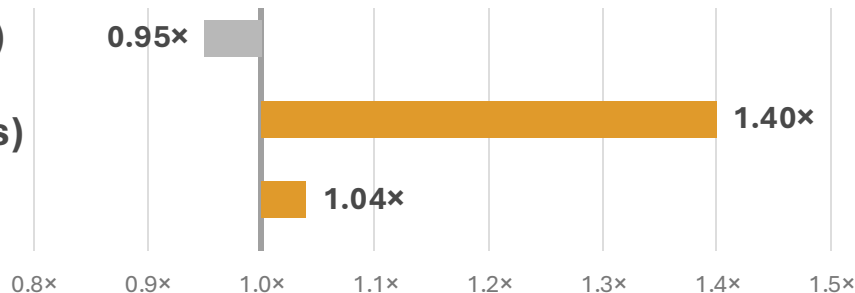
Relative speed-up vs. Megatron

step time · higher is better

GLM-4.5-Air-106B-A12B (32 GPUs)

MiniMax-M3-428B-A23B (32 GPUs)

GLM-5.2-744B-A40B (64 GPUs)



Speed-up = Megatron step time ÷ DeepSpeed step time.

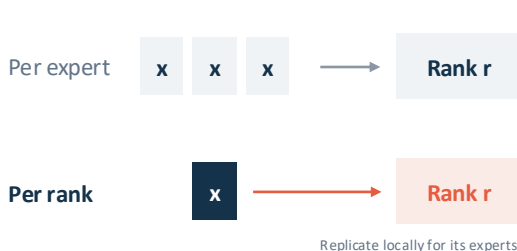
AutoEP Optimization: Fast MoE, End to End

Move less data. Fuse expert work. Make communication and compute overlap.

Expert dispatch

01 Move less data

DEEPEP-ENABLED DISPATCH



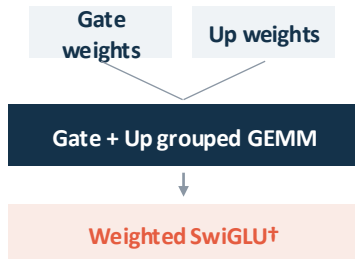
Send once per destination rank.

DeepEP transport + backend-owned dispatch and expert layout.

Expert compute

02 Fuse the expert path

GATE+UP GEMM · WEIGHTED SWIGLU



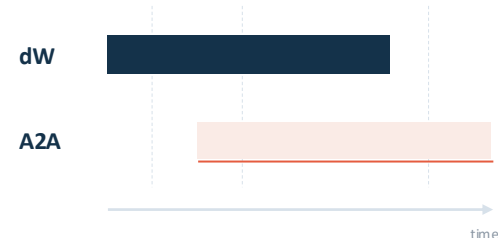
Pack projections. Fold in weighting.

Remove a separate row-weighting pass; keep the original parameter layout.

Backward scheduling

03 Make communication overlap with compute

dW / A2A · SM-AWARE SCHEDULING



Give communication room to run.

Defer dW, overlap with A2A, and reserve SMs for communication.

Speedups Beyond the MoE block

Fewer launches. Less replay. More memory headroom.

Kernel & compiler optimization

01 Fuse small kernels

RMSNORM / ROPE · REGIONAL COMPILE

Kernel fusion



Or: regional compile · native path



Fewer launches and HBM passes.

Fuse RMSNorm / RoPE—or compile around the eager MoE boundary.

Checkpoint-aware execution

02 Stop recomputing attention

ATTENTION OUTPUT STASH



Keep the result. Skip core replay.

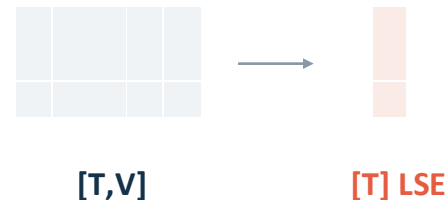
Trade selected activation memory for less checkpoint recomputation.

Memory-efficient training

03 Releasing FP32 copies of logits

MEMORY-LEAN CAUSAL-LM LOSS

Extra FP32 loss state



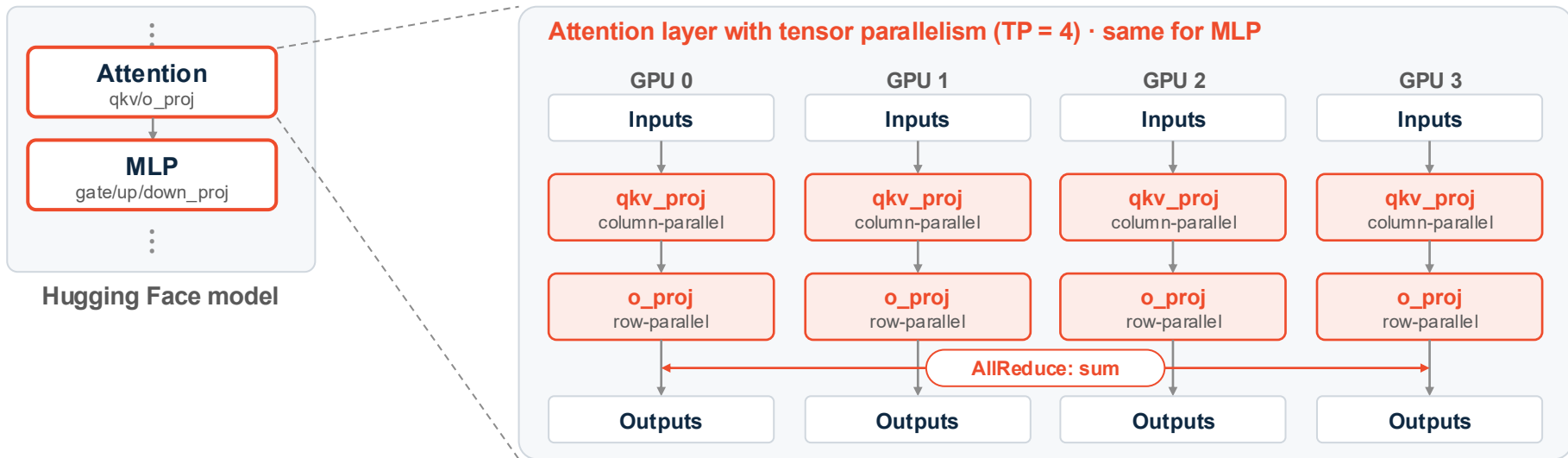
Keep logits—not FP32 copies.

Save LSE; rebuild probabilities only when backward needs them.

AutoTP (Tensor Parallel)

with **Guokai Ma**, **Mingzhi Liu**, and many others!

- **Tensor parallelism (TP)**: splits each weight matrix across GPUs
- **Detect and replace**: Swaps in column-/row-parallel layers



TP Plans Without Touching Model Code

- Every framework has to detect attention and MLP layers and rewrite them
- The differences are:
 - Where the layout comes from
 - What you touch to change it

Capability	DeepSpeed AutoTP	HF transformers	Megatron	TorchTitan
① Where the layout comes from which layers are column / row	HF <code>tp_plan</code> or regex rules in config	<code>tp_plan</code> from the model config	Megatron-Core model specs in code	<code>parallelize plan</code> per model, in code
② Custom layout without touching model code	✓ Config regex patterns → col / row	✓ Config <code>tp_plan</code> dict	Code a bridge class	Code edit the plan
③ Hugging Face checkpoints no offline conversion	✓ HF model as-is	✓ native	✓ Bridge loaded on the fly	✓ Adapters <code>initial_load_in_hf</code>

Benchmark: Single-Node Dense TP

Model: Qwen3-32B (dense)
HW: 8× H100
Seq len: 8,192
Parallel: TP8
Batch: $\mu 1$ · GBS1
Recompute: full AC

On par with Megatron

- **0.78 s vs. 0.81 s per step**
- **1.7× over plain AutoTP with DeepCompile**

Mean time per step

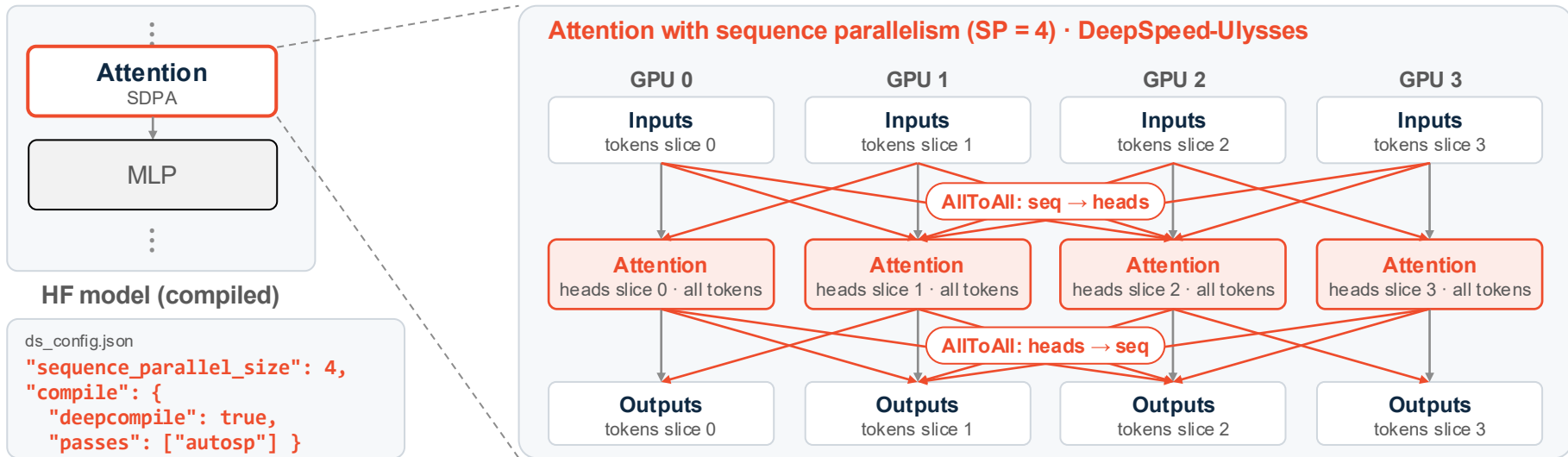
seconds · lower is better



Mean time per step, same model and setup for all four rows.

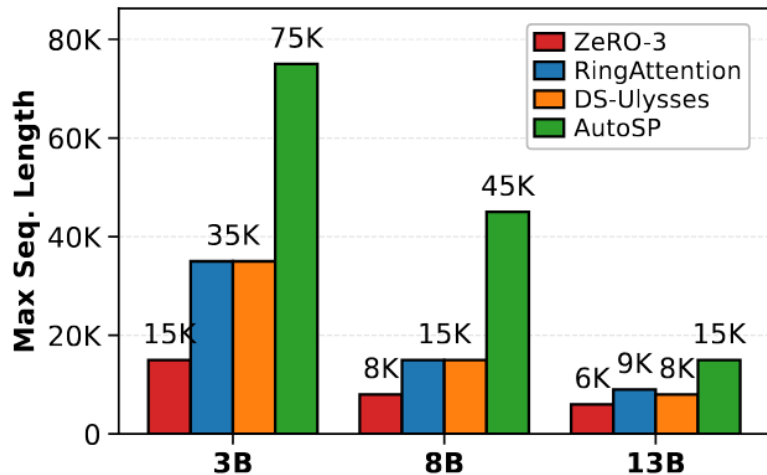
AutoSP (Sequence Parallel)

- **Sequence parallelism (SP)**: splits one long sequence across GPUs to train longer contexts
- **Compile and rewrite**: DeepCompile pass finds the attention calls in the computation graph and inserts the all-to-all. Ulysses, generated automatically

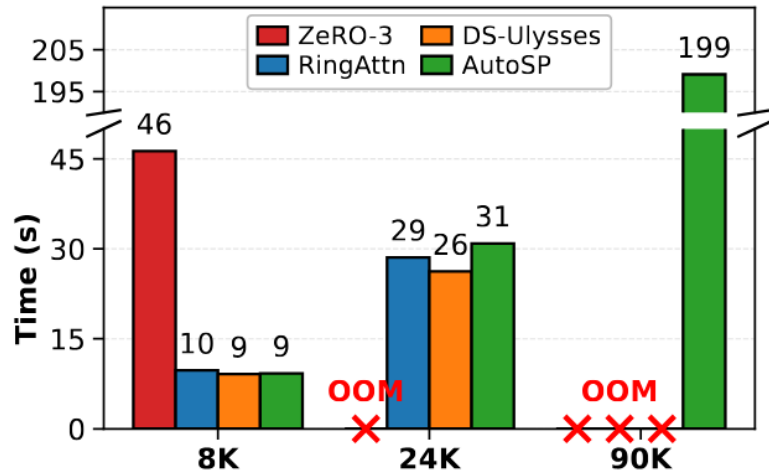


Benchmark

- **Up to 3× longer context** than DeepSpeed-Ulysses / RingAttention at the same step time (8B: 45K vs. 15K tokens, 9 s per step at 8K)



Max trainable sequence length — higher is better



Time per step (s) — lower is better

Setup: Llama 3.1 models on 8× A100-80GB, PyTorch 2.7; baselines (torch-compiled): ZeRO-3, RingFlashAttention, DeepSpeed-Ulysses. Figure: PyTorch blog “Introducing AutoSP” (May 2026).

AutoSP (Sequence Parallel)

<https://pytorch.org/blog/introducing-autosp/>



Learn ▾ Community ▾ Projects ▾ Docs ▾ Blog & News ▾

with **Ahan Gupta, Zhihao Wang, Neel Dani, Olatunji Ruwase, Minjia Zhang**

Blog

Introducing AutoSP

By Ahan Gupta¹, Zhihao Wang¹, Neel Dani¹, Masahiro Tanaka², Olatunji Ruwase³, Minjia Zhang¹ | April 29, 2026

¹ SSAIL Lab, University of Illinois Urbana-Champaign, ² Anyscale, ³ Snowflake

TL;DR: AutoSP automatically converts standard transformer training code into sequence-parallel code for long-context LLM training across multiple GPUs. Integrated with DeepSpeed, it increases maximum trainable context length with little runtime overhead versus hand-written baselines.

Featured
projects



DeepSpeed

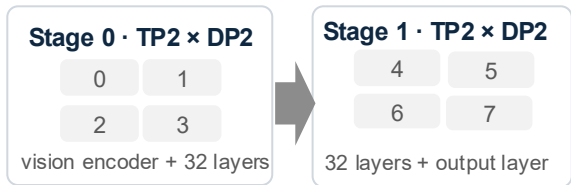
Gupta, Wang, Dani, Tanaka, Ruwase, Zhang. “AutoSP: Unlocking Long-Context LLM Training via Compiler-Based Sequence Parallelism.” ICLR 2026.

Ray-based Pipeline Parallel

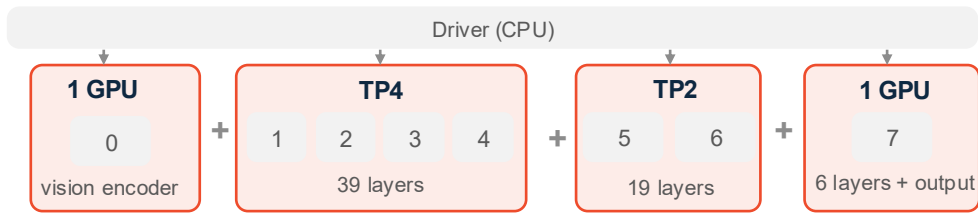
with Naveenraj
Kamalakkannan

- **Per-stage layouts:** each pipeline stage can use a different number of GPUs and different parallelisms (TP / DP / SP / EP / ZeRO).
- **Ray for allocation and wiring:** Ray assigns GPUs to the stages and connects them

Conventional PP



Ray-based PP



Ray-based Pipeline Parallel

Heterogeneous PP

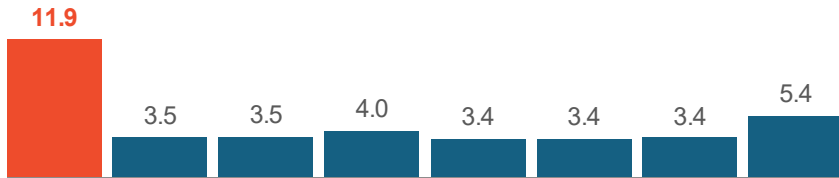
- Vision encoder and LLM are different workloads, so no single parallel layout fits both
- Hence per-stage parallelism, which Megatron-MIMO (NVIDIA) also implements

Partitioning by measured time, not by layer count

- Imbalanced stages make a pipeline wait on its slowest stage.

Qwen3.5-35B-A3B on 16× H100 · measured busy seconds per stage, per step · bar width = GPUs in the stage

Conventional PP · 8 stages × 2 GPUs



Stage 0 also runs the vision encoder: **11.9 s** against 3.4–5.4 s.
Every stage waits on it.

Ray-based PP · vision stage of 4 GPUs + 6 stages × 2



Vision encoder on its own 4-GPU stage: every stage 3.7–4.9 s.
Step **6.38 s**, 1.50× the 8-stage pipeline (9.60 s).

Benchmark: VLM Pipeline, Single Node

Model: Qwen3.5-27B (VLM)
HW: 8× H100 80GB
Seq len: 2,048
Batch: 64 seq / step
Data: CORD-v2 receipts
Config: best per framework

1.28× Megatron-Bridge, 1.18× TorchTitan

- Same 8-stage cut as Megatron-Bridge
- All three recompute every layer

Throughput per GPU

tokens/s · higher is better

DeepSpeed Ray-based PP

PP8, measured cut · 11.19 s/step

1,465

TorchTitan

FSDP over 8 GPUs · 13.16 s/step

1,245

DeepSpeed 1.18× faster

Megatron-Bridge v0.6.0

PP8, Ray-based PP's cut · 14.34 s/step

1,143

DeepSpeed 1.28× faster

Step time: median of steps 6–50, one run each. Same data, loss and optimizer. Megatron-Bridge's own even split ran out of memory.

Benchmark: VLM MoE Pipeline, 2 Nodes

Model: Qwen3.5-35B-A3B (MoE)
HW: 16× H100 (2 nodes)
Seq len: 2,048
Batch: 128 seq / step

1.36× Megatron-Bridge, 1.43× TorchTitan

- **Vision encoder on its own 4-GPU stage**
- **1.50× over our own even 8-stage cut**

Throughput per GPU

tokens/s · higher is better

DeepSpeed Ray-based PP

vision 4 GPUs + 6 stages × 2 · 6.38 s/step

2,567

Megatron-Bridge v0.6.0

TP1 × PP2 × EP8 (NVIDIA recipe) · 8.67 s/step

1,890

DeepSpeed 1.36× faster

TorchTitan

FSDP over 16 GPUs + EP8 · 9.11 s/step

1,798

DeepSpeed 1.43× faster

DeepSpeed Ray-based PP

PP8 × DP2, measured cut · 9.60 s/step

1,707

vision stage 1.50× faster

Step time: median of steps 6–50, one run each. Same data, loss and optimizer. Only the first row recomputes every layer.

Benchmark: vs. Megatron-MIMO

Models: Qwen3.5 9B/27B/35B
HW: H100 80GB
Layout: MIMO's own
Vision: on its own GPUs

Within 7% of Megatron-MIMO

- **0.93–1.04×** on its own layouts (9B, 27B)
- **35B-A3B MoE: MIMO does not run**

Relative throughput vs. Megatron-MIMO

Qwen3.5-9B · 5 GPUs

vision 1 GPU + TP2 × PP2 · 2,798 vs. 2,680

Qwen3.5-9B · 5 GPUs

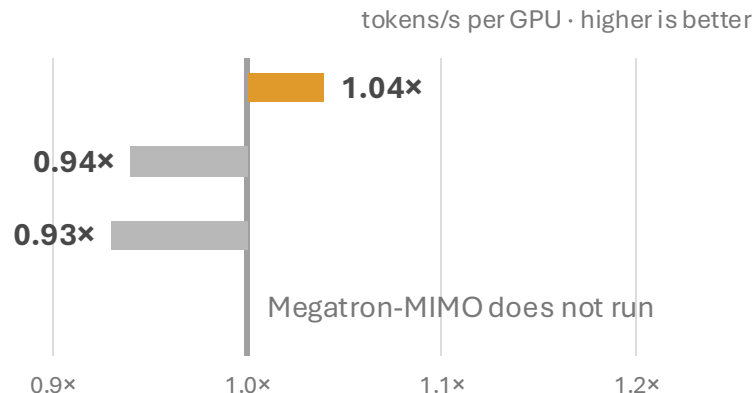
vision 1 GPU + TP2 × DP2 · 3,124 vs. 3,331

Qwen3.5-27B · 9 GPUs

vision 1 GPU + TP4 × PP2 · 921 vs. 994

Qwen3.5-35B-A3B · 16 GPUs

vision on its own GPUs · DeepSpeed 2,567



Median of steps 6–50 (27B: 4–10). 9B: MIMO's fixed settings on both. 27B: bf16 gradients and a 2-byte master copy on both.

Acknowledgements

AutoEP

Han Yu (DeepEP transport, fused kernels, regional compile) · **Peng Du** (kernels, ZeRO-3 folding, sparse attention for MiniMax / GLM) · **Hongwei Chen** (Triton grouped GEMM, routing) · **@jinyouzhi** (checkpoint placement) · **@hazelduan** (HiFloat8 training) · **@0z5a** (folding fixes) · **@alanhuangyoo** (gradient clipping) · **@gss10282023** (gradient clipping) · **Goransh Walia** (gradient clipping) · **@Silas-Zeng** (routing validation) · **Vineeth Sai Varikuntla** (fixes) · **@poorpaper** (router bias buffers) · **@Achyuthan-S** (shard restore)

AutoTP

Mingzhi Liu (AutoTP training, async all-reduce) · **Guokai Ma** (HF tp_plan support, model coverage) · **Yejing Lai** (DeepSeek / Qwen3 support) · **@jinyouzhi** (uneven sharding, universal checkpoints, vocab-parallel head) · **Naveenraj Kamalakannan** (compiler pass) · **Peng Du** (AutoEP + AutoTP folding) · **@0z5a** (checkpoint resume, output head) · **@Achyuthan-S** (affine checkpoint maps) · **@nathon-lee** (universal checkpoint) · **Zhipeng Wang** (ZeRO-3 fix) · **@ebarkhordar** (tp_plan embeddings) · **Zupeng Wang** (metadata fix) · **@gaoxiaomo** (lm_head routing) · **@promptsmith1990** (Qwen3.5 norms) · **Lun Zhongwang** · **@ranzhejiang** (Qwen3-MoE loading) · **@HollowMan6** (fixes) · **Hongwei Chen** (tests) · **@alanhuangyoo** · **@mayuriphad** (checkpoint loading) · **GM You** (AutoTP for MoE)

AutoSP

Ahan Gupta (AutoSP author) · **Neel Dani** (AutoSP author) · **Minjia Zhang** (lead) · **Zhipeng Wang** · **Olatunji Ruwase** · **@jinyouzhi** (loss aggregation, mesh validation) · **@nathon-lee** (multimodal support) · **Peng Du** (sequence sharding) · **Ankit Dhandharia** (fixes) · **@Zhuoxi2000** (fixes) · **Parth Vadera** (fixes)

Pipeline Parallel

Naveenraj Kamalakannan (Ray-based pipeline engine)

Summary

DeepSpeed now shows matching performance with other SoTA frameworks

EP

AutoEP

TP

AutoTP

SP

AutoSP

PP

Ray-based



Join our Slack!

Not the DeepSpeed you remember

ZeRO-only → **TP, EP, SP and PP** at state-of-the-art performance

Same ZeRO-level usability: one config, no model code changes

*Not as battle-tested as Megatron yet. But solid enough to be **your starting point** for optimization.*

Build it with us

- Try it on your model. Tell us what is slow or broken.
- Contributions and collaborations are welcome
- github.com/deepspeedai/DeepSpeed

If you care about performance: try it, break it, and build it with us.