

Integrating TorchTPU into TorchTitan

and Optimizing Models for TPU



Google

Aleksey Vlasenko

TLM, Pytorch TPU training team



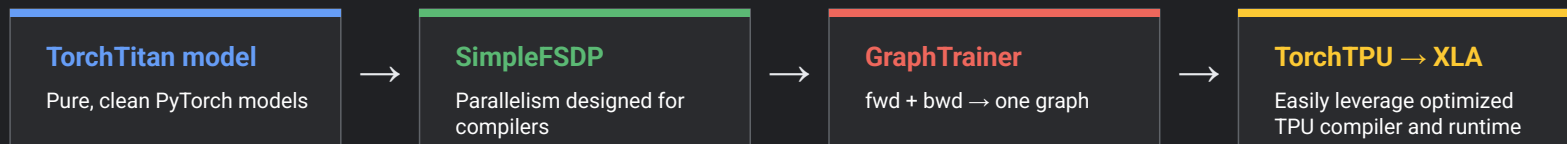
Meta

Will Constable

Engineer, PyTorch Distributed

Why TorchTitan Is a Great Fit for TPU

Built to demonstrate pre- and post-training at scale, from pieces that map cleanly onto TPU.



Easy to retarget to TPU

No CUDA-specific code to remove.

Models and configs are modular, it's easy to swap in TPU-optimized modules or kernels per recipe.

Collectives are Traceable

SimpleFSDP and DeviceMesh represent parameter sharding as traceable collectives in the captured graph.

Configs let recipes choose optimal parallelism setup for TPU, without changing the Model.

Flexible Train Step Capture

make_fx captures forward, loss and backward ahead of time.

Graph passes choose what to save, recompute or offload.

Seamless interoperability with eager ops and localized torch.compile usage.

Inherits XLA's TPU tuning

Lowered to the same compiler JAX uses on TPU.

Collective overlap and Offload/Reload scheduling are handled by XLA and offloaded to SparseCores.

Pallas kernels plug in easily, and PyTorch operators get XLA fusion.

What We Built on TorchTitan

We forked TorchTitan to move fast, and built mostly through its existing extension points.

Upstream TorchTitan

Our fork: github.com/google-pytorch/torchtitan

Model blocks & kernels

GroupedExperts
SDPA · cross_entropy



TpuGroupedExperts · experiments/tpu/kernels

MegaBlox grouped-matmul experts with a fast token combine, Tokamax Splash attention, and a fused linear + cross-entropy loss. Swapped in by config, no model edits.

Parallelism

ParallelDims



enable_tpu_topo_mesh

Mesh ranks follow the physical torus, so collectives take libtpu's fast multi-ring path: about 4× the reduce-scatter bandwidth.

Trainer

GraphTrainer



TpuGraphTrainer(GraphTrainer)

Compiles the captured step with XLA instead of Inductor. TPU passes keep PyTorch's AC and FSDP memory plan intact, and scan or graph splitting handles graphs too big to compile whole.

Config

Trainer.Config

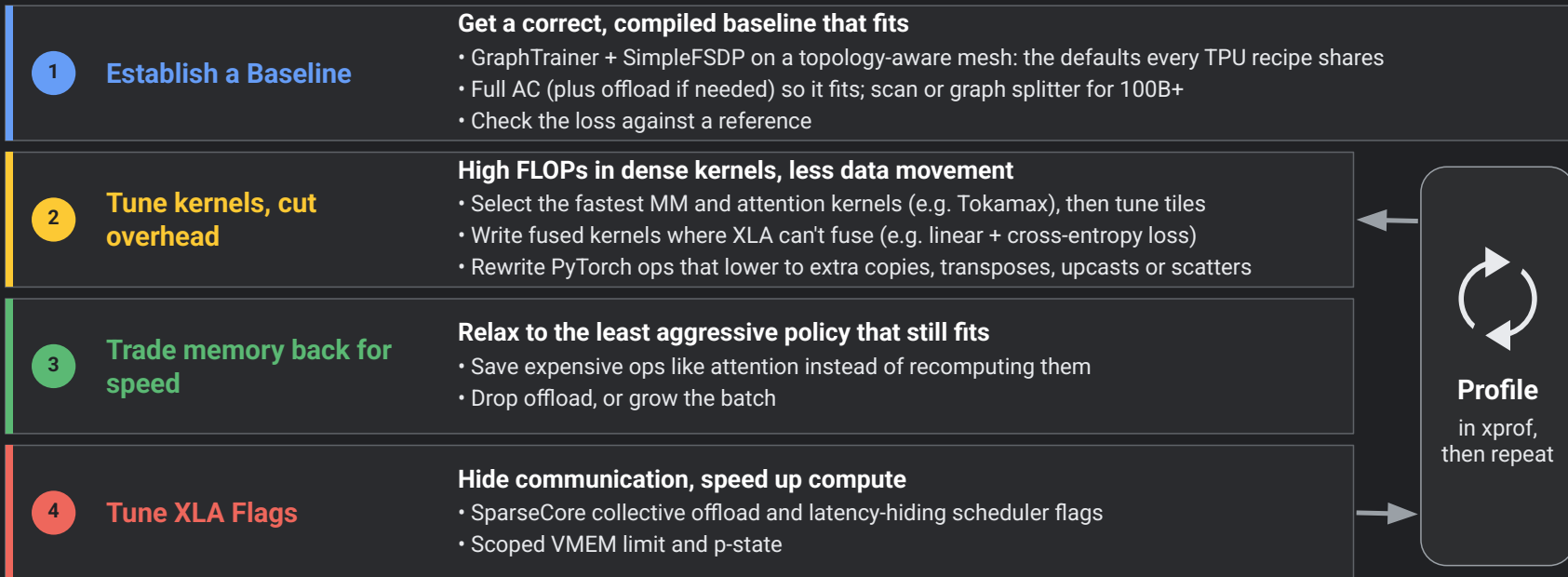


TPUTrainerConfig · recipes/

Tuned recipes: Qwen3 8B / 32B / 30B-A3B / 235B-A22B, Llama 3 8B / 70B, FLUX.1, Conformer.

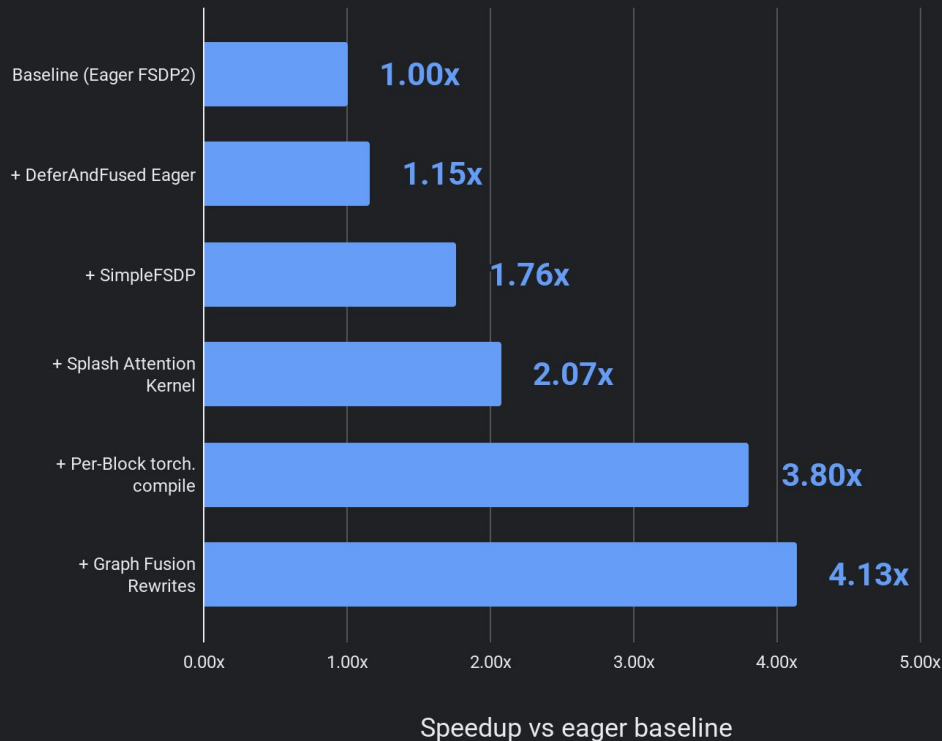
How to Optimize a Model on TPU

Build on top of our shared TorchTitan infrastructure, and tune your model's performance



Applying optimization recipes: the FLUX model case study

FLUX.1 on TPU v6e, same hardware and batch: 4.13x over the eager baseline, 5.7% -> 23.7% MFU, 11.49 s -> 2.78 s per step



Baseline 11.49 s, 5.7% MFU

Eager FSDP2 with full activation checkpointing: one XLA program per op, collectives issued from Python.

+ DeferAndFused eager 1.15x, 6.6% MFU

TorchTPU defers dispatch and fuses runs of eager ops into larger XLA programs; fewer launches, same code.

+ SimpleFSDP 1.76x, 10.1% MFU

Sharding as traceable collectives instead of FSDP2 hooks, so XLA schedules and overlaps the all-gathers.

+ Splash attention 2.07x, 11.9% MFU

Pallas flash-attention kernel in place of SDPA; attention stops dominating the step.

+ Per-block torch.compile 3.80x, 21.8% MFU

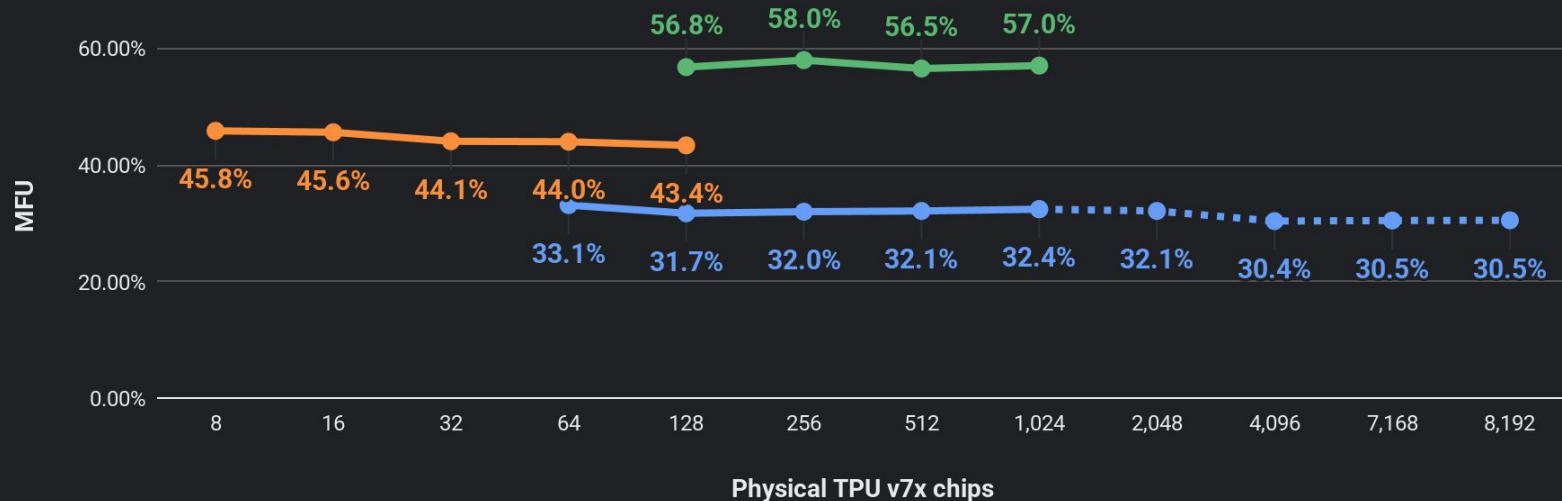
Each transformer block becomes one compiled XLA program: whole-block fusion, no per-op dispatch.

+ Graph fusion rewrites 4.13x, 23.7% MFU

Rewrite the ops that lowered badly (copies, upcasts, scatters) so XLA fuses them; found in xprof.

It Scales: Same Code From 8 to 8,192 Chips

Qwen3-235B, Llama3 405B, Muse Glimmer 30B on TPU v7x; MFU per chip; fixed per-chip batch, so global batch grows with chips



Muse Glimmer 30B, 8 -> 128 chips

95% scaling efficiency: 6,096 -> 5,768 TPS/chip at sequence 16k; same loss at every size.

Qwen3-235B, MoE, 64 -> 8,192 chips

92% scaling efficiency: 4,568 -> 4,208 TPS/chip across 128x more chips.

..... **weak scaling:** multi-slice HSDP over DCN with a fixed per-chip batch (beyond ~1k chips a scaling experiment, not a training recipe).

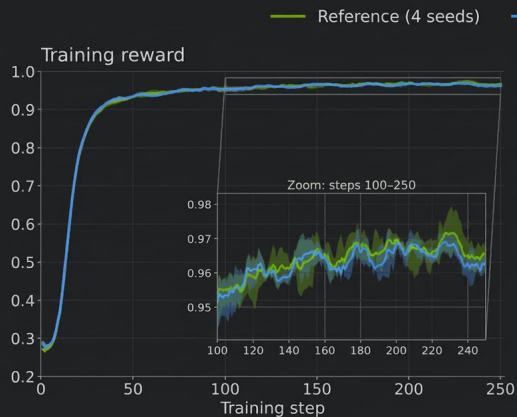
Llama3 405B, 128 -> 1,024 chips

Linear scaling efficiency: 249 -> 250 TPS/chip across 8x more chips; dense model, ~57% MFU at every size.

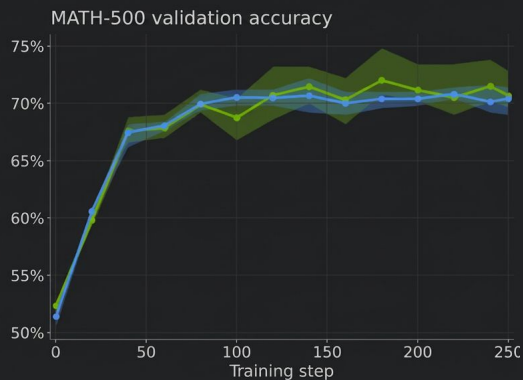
verl on TPU: GRPO on Cloud TPU

Proof of concept training Qwen3-4B-Base on verl TPU, and validating model quality against reference

Qwen3-4B-Base · verl GRPO · 250 steps · same recipe and hyperparameters on TPU v6e

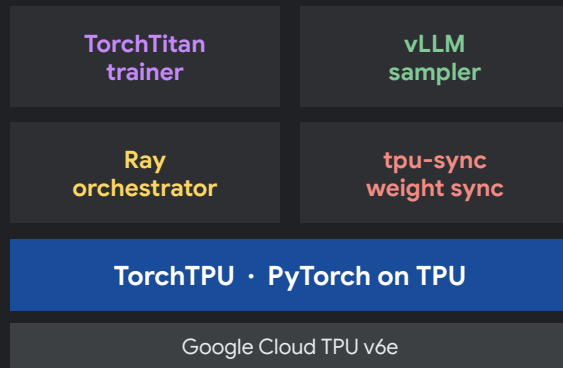


Training reward parity



MATH-500 parity at step 250

verl stack on TPU



Recipe (verl + TPU plugin)

verl/examples/tpu/grpo

TorchTitan Proves TorchTPU Is Ready for Production at Scale

TorchTitan is where TorchTPU proves itself for training: production-scale models, unchanged PyTorch code, single host to multi-pod. Everything we used to get there is yours to use: github.com/google-pytorch/torchtitan.

Two ways in: adopt TorchTitan as your training framework, or keep your training code and take the pieces you need.

Try it: run a recipe

- **Llama 3, Qwen3, FLUX.1, Muse Glimmer, Mamba2, Conformer** on v6e and v7x
- **Launch block and libtpu flags** ship with every recipe; single-host TPU VM or multi-host GKE
- **Known performance:** step time, TPS, MFU and peak HBM stated per recipe, with what each flag buys

Adopt it: bring your own model

- **HF Transformers backend:** validated on 500+ decoder-style LLMs; one stock TPU config, no model-code change
- **Model-onboarding skill** for engineers and coding agents: the same ladder, with tests, acceptance bars and traps built in
- **Optimization guide** for the last percent

Take what you need

- **Kernels are standard PyTorch custom ops:** splash attention, fused CE, grouped matmul, route-score bwd, Mamba SSD
- **SimpleFSDP and the Graph Trainer** are modules you import
- **libtpu flags and the playbook** apply to any TorchTPU workload; recipes double as documentation

Roadmap

Production

ML Goodput measurement; Multi-Tier Checkpointing on GKE

Precision

float8 recipe from Llama 3 70B to the full model set

Hardware

TPU 8t and 8i as they reach Google Cloud, same recipes

Parallelism

expert parallelism for MoE kernels, then long context

Models

Qwen3.8-Max 2.4T MoE; Kimi K3 2.8T, Kimi Delta Attention

Start with a recipe, then bring your own model.

Start: torchtitan/experiments/tpu/README.md | Issues and PRs: github.com/google-pytorch/torchtitan

