



CONFERENCE

—NORTH AMERICA 2026—

TorchTitan GraphTrainer

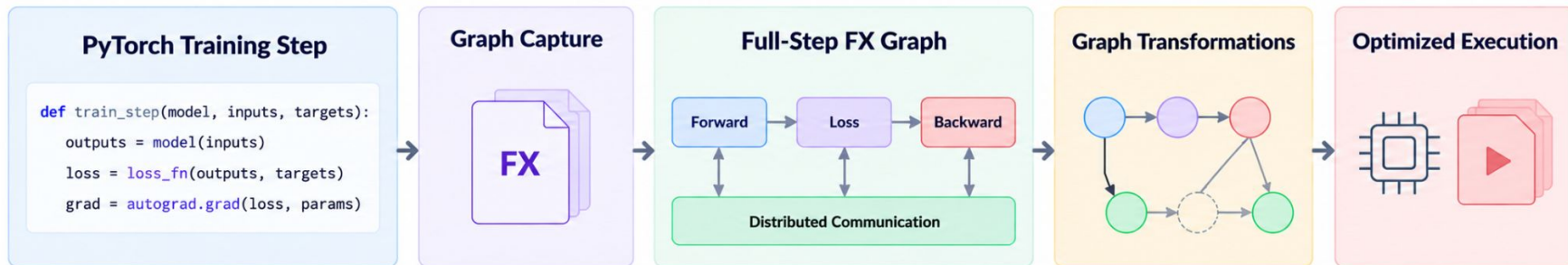
A Graph-Based Approach to Distributed LLM Training

Sherlock Huang, Sanket Purandare

Joint work with wider PyTorch team @ Meta

What is GraphTrainer?

A graph-based execution & optimization framework for distributed LLM training



Capture Everything · Optimize Globally · Execute Efficiently

Why Graph-based Training?

Hardware Heterogeneity

One graph, many backends

ASICs need a graph

Further accelerate GPU-like backends

Operational Simplicity

Decouples Modeling from Optimization

Composable graph passes

Efficient substrate for agentic hillclimb

Performance

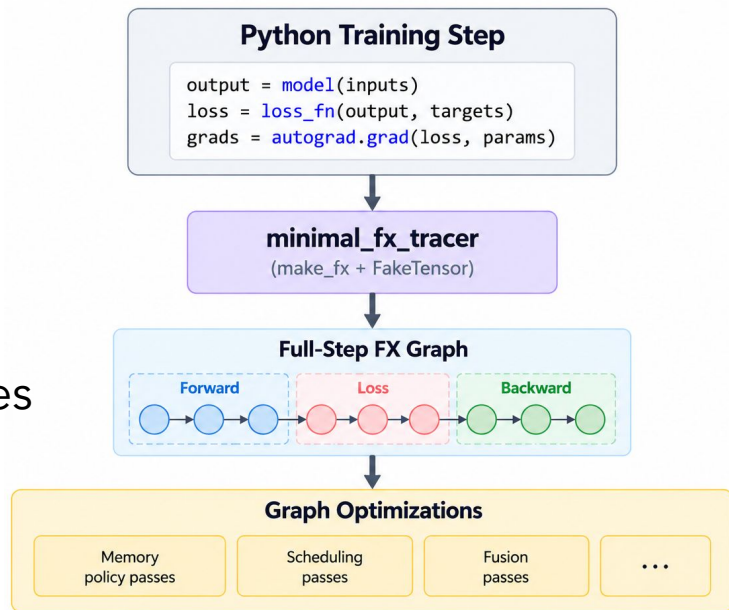
Optimize beyond Eager

Fine-grain comp/comm overlap

Cross-module and backward fusions

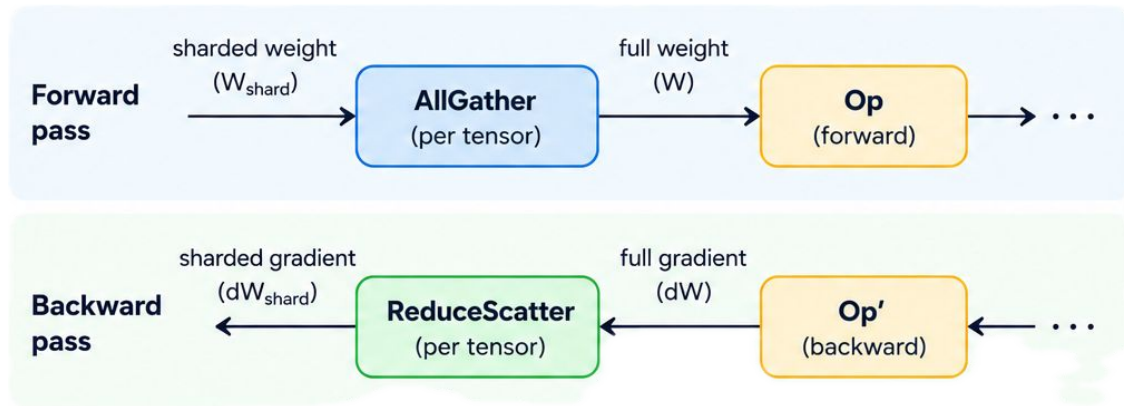
Minimal FX Tracer: Capturing The Full Training Step

- `make_fx`-based tracing with FakeTensor
- Full-step capture: forward-loss-backward, communication in graph
- Explicit state: params/buffers lifted as graph inputs
- `fx.traceback.annotate` for tagging nodes
 - 'my_tag': 'any_value' in node.meta
 - Help graph pass finding nodes



SimpleFSDP: Making FSDP Graph Friendly

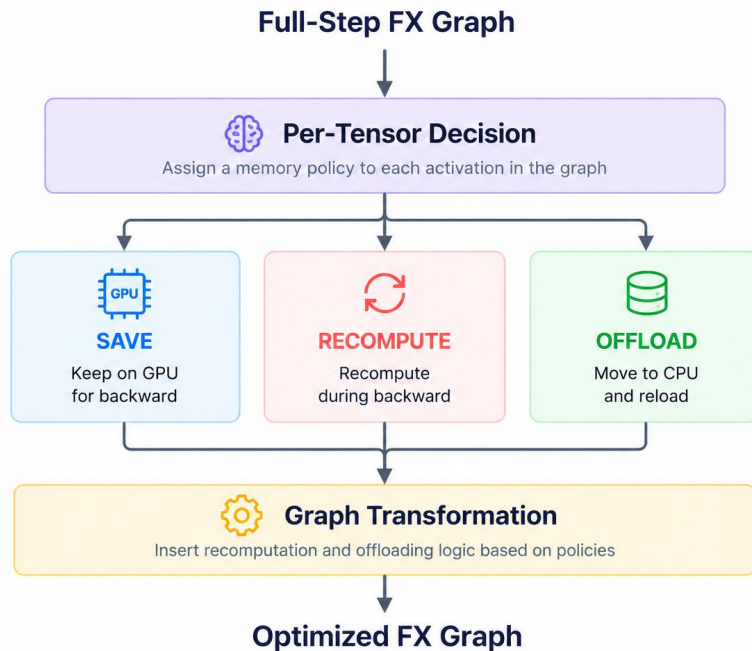
- Simpler Infra: avoid complex hook-based orchestration
- Weight allgather and gradients reduce-scatter as traceable ops
- FSDP bucketing and prefetching implemented as graph pass



Communication becomes part of the graph—not hidden behind module hooks.

Unified Per-Tensor Memory Management

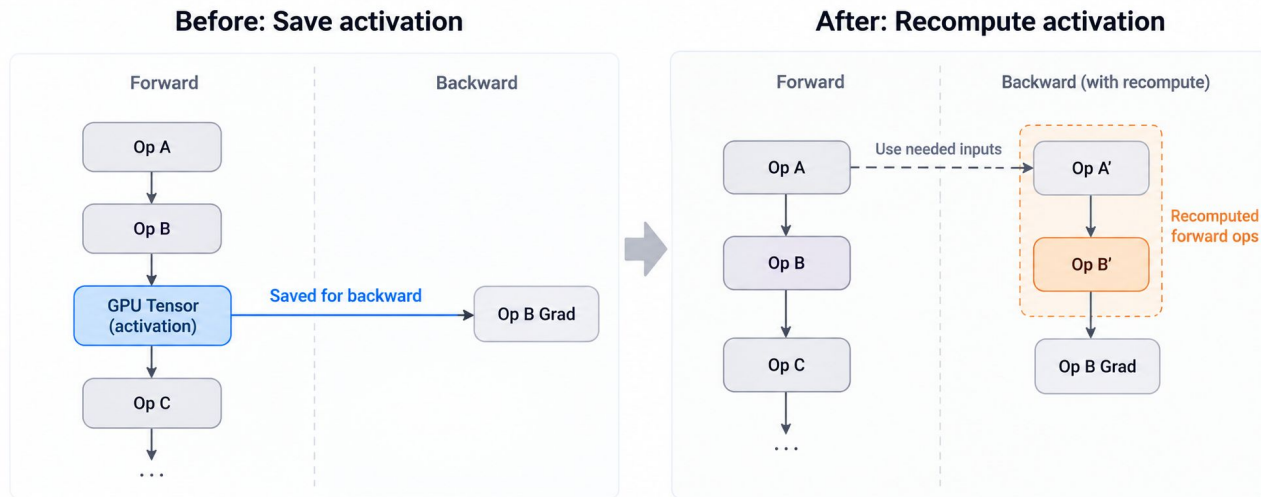
- Per-tensor flexibility: No restriction to module-level AC boundary
- Extensible registry for custom memory policy
- Recompute and Offload are implemented as graph passes



Graph-based Activation Checkpointing

Duplicate selected region in backward

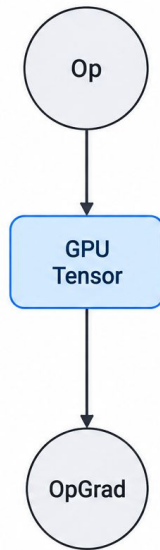
Rewire backward dependencies to reduce activation memory



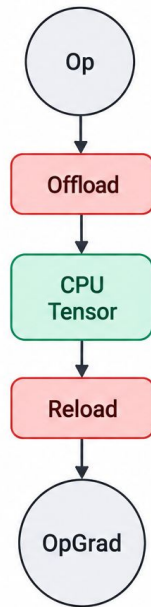
CPU Offload via Graph Transformation

Insert offload and reload
Schedule to overlap data transfer with
computation

Before: Keep on GPU



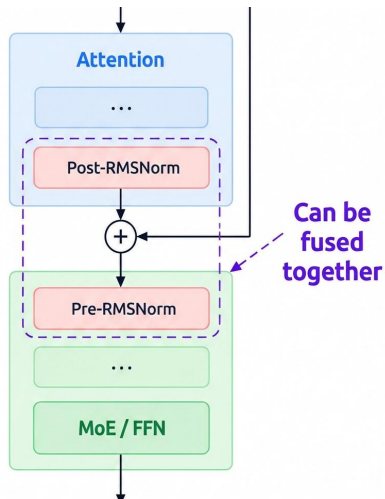
After: CPU Offload



Fusion beyond Eager Boundaries

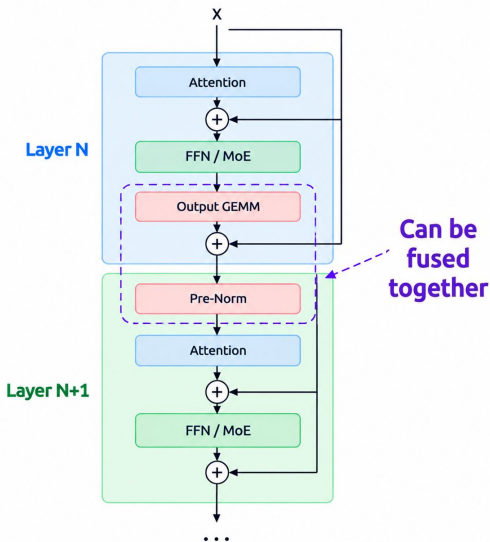
Cross Module Fusion

Attention post-norm + residual +
FFN pre-norm



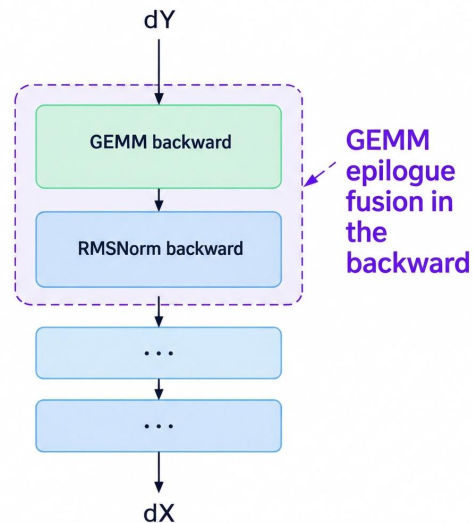
Cross Layer Fusion

FFN output Gemm + residual +
next-layer pre-norm



Fusions in Backward

CODA-style Gemm epilogue



Unlocking a long tail of fusion opportunities that are difficult to achieve in eager.

Regional Inductor Compilation

Selective Compilation

- `fx.traceback.annotate` for marking regions to compile

Numerical Control

- User opt-in for bitwise changing compilation

```
1 with fx.traceback.annotate(  
2     {"compile_with_inductor": True}  
3 ):  
4     x = opA(x)  
5     x = opB(x)  
6  
7 x = opC(x)
```



Transformed FX Graph (after selective compilation)



Compile where you want. Preserve numerical behavior where you need.

Scheduling Optimizations

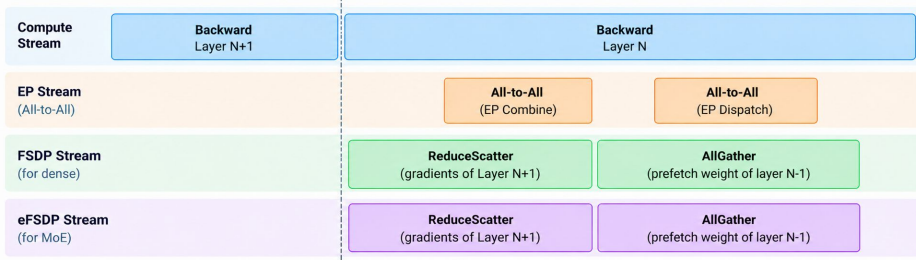
Global visibility of all communication

Scheduling == moving comm nodes around in graph

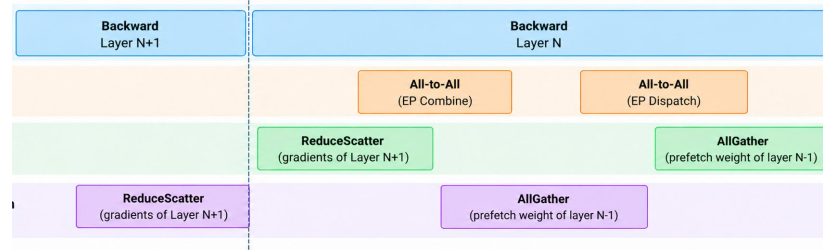
- Topology order guarantees data dependency

Contention-aware scheduling, without the limitation of module boundary hook

Before Optimization
(communication overlapped, contention)



After Optimization
(contention-aware, globally scheduled)



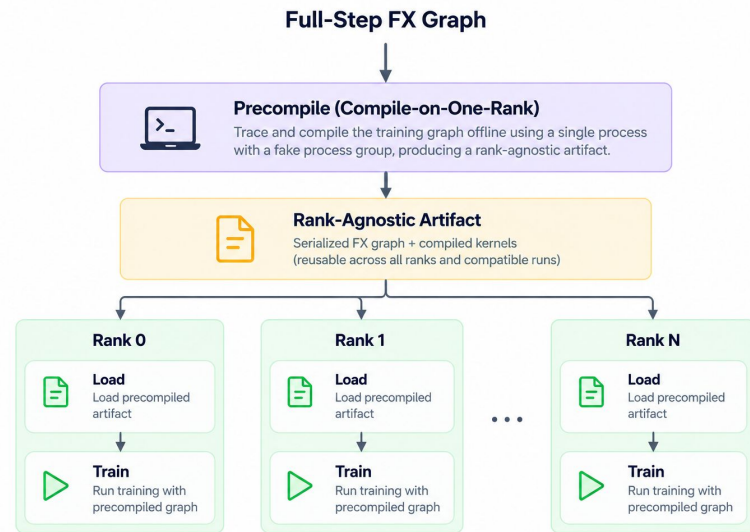
Precompile Once, Run Anywhere

Fast startup

- Avoids redundant tracing and compilation
- All ranks loads the precompiled artifact

Compile-on-one-rank

- Represent rank-dependent behavior symbolically
- Compile without needing full cluster



GraphPP: Graph-based Pipeline Parallelism

Eager pipeline parallelism

Limited control over backward

Full flexibility requires a manual backward.

Thread and stream orchestration

Fine-grained overlap, such as DualPipe, requires explicit orchestration.

Composability challenges

Branching model code interacts with activation checkpointing and torch.compile.

GraphPP

Full control over backward

Functional graph capture and AOTAutograd.

No thread management

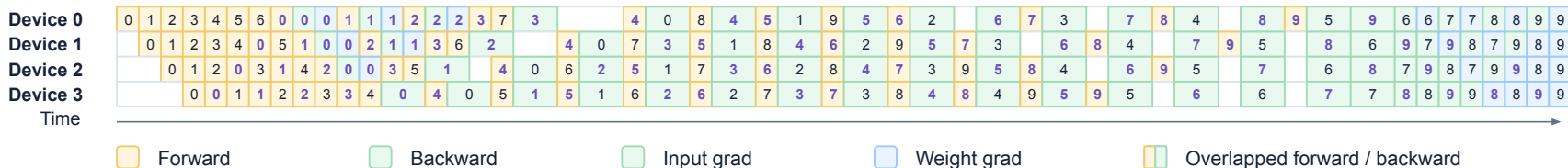
Overlap can run off the main thread.
Stream management is optional.

Compiler-first

No model changes once traceable.
Fine-grained activations and Inductor performance.

GraphPP Schedule IR

A schedule is a sequence of graph actions



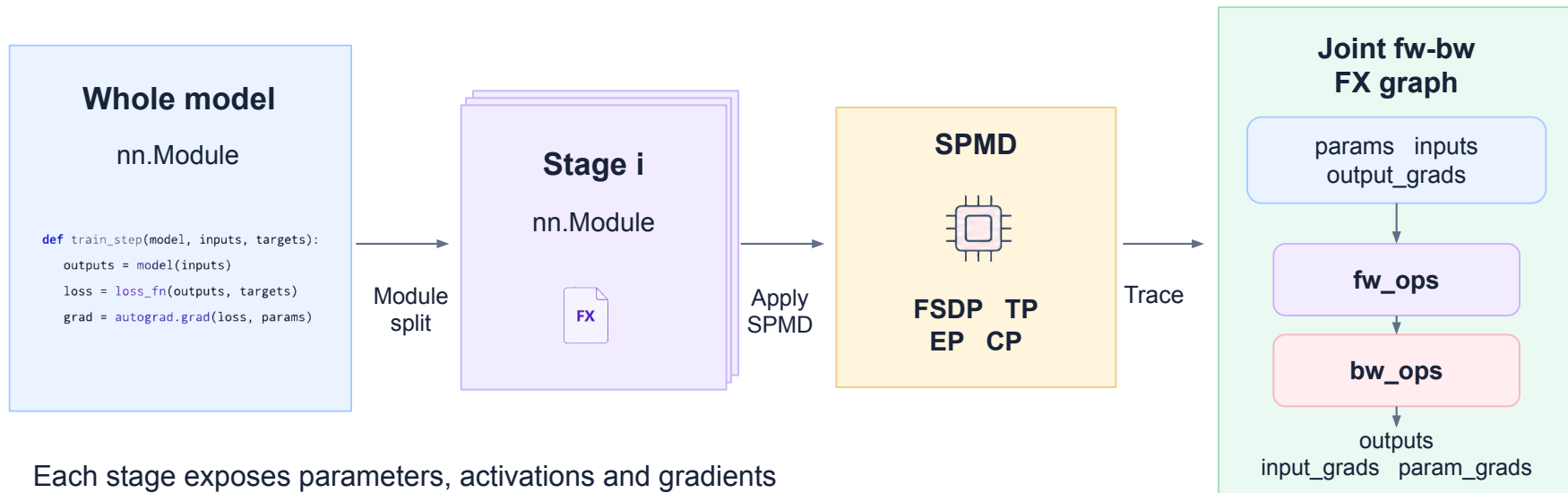
Compute actions

iFj	Stage i forward, microbatch j
ilj	Stage i input gradient, microbatch j
iWj	Stage i weight gradient, microbatch j
iBj	Stage i full backward, microbatch j
(iFj; mBk)	Stage i forward and stage m backward with microbatches j and k

State and communication

Unshard i	Unshard stage i parameters
Reshard i	Reshard stage i parameters
ReduceGrad i	Reduce stage i parameter gradients
iSEND_F/Bj	Send output / input_grad to next PP rank
iRECV_F/Bj	Receive input / output_grad from previous PP rank

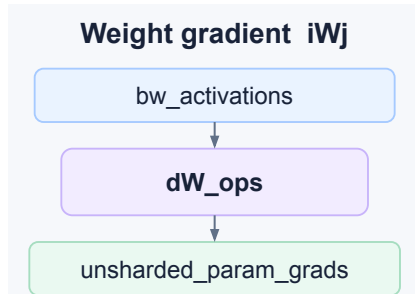
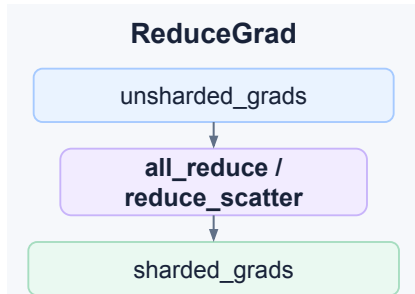
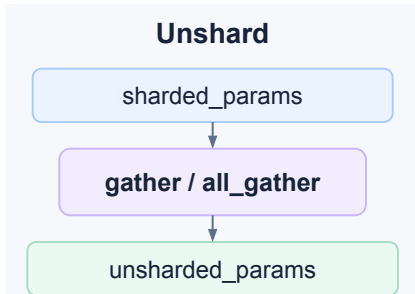
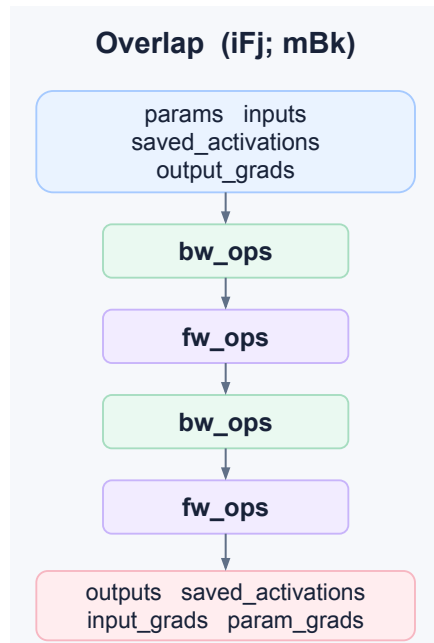
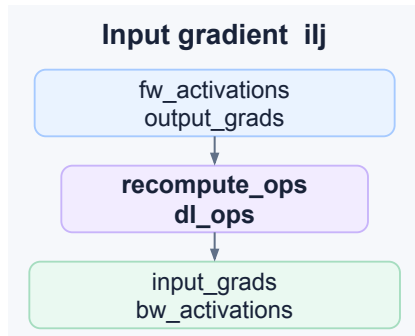
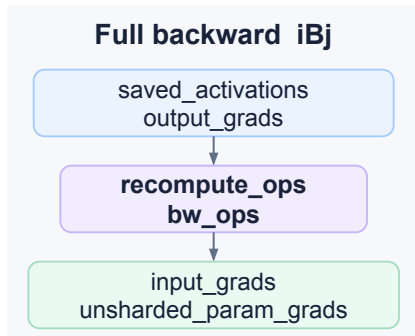
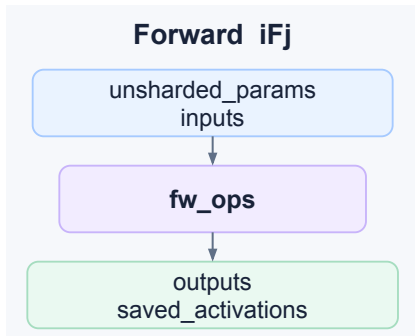
GraphTrainer for GraphPP



Each stage exposes parameters, activations and gradients through a functional graph.

Schedule IR Actions and Graphs

Forward and full backward graphs exclude FSDP collectives



Graph-based PP Runtime



Per-stage (i) graph callables

unshard
fw full_bw
bw_dl bw_dW
reduce_grad

Across stages (i, m)

overlap_fw_bw

Schedule IR actions

iFj
iIj
iWj
iBj
(iFj; mBk)
Unshard i
Reshard i
ReduceGrad i
iSEND_F/Bj
iRECV_F/Bj

Per microbatch (j)

Receive buffers

inputs
output_grads

Send buffers

outputs
input_grads

Activation cache

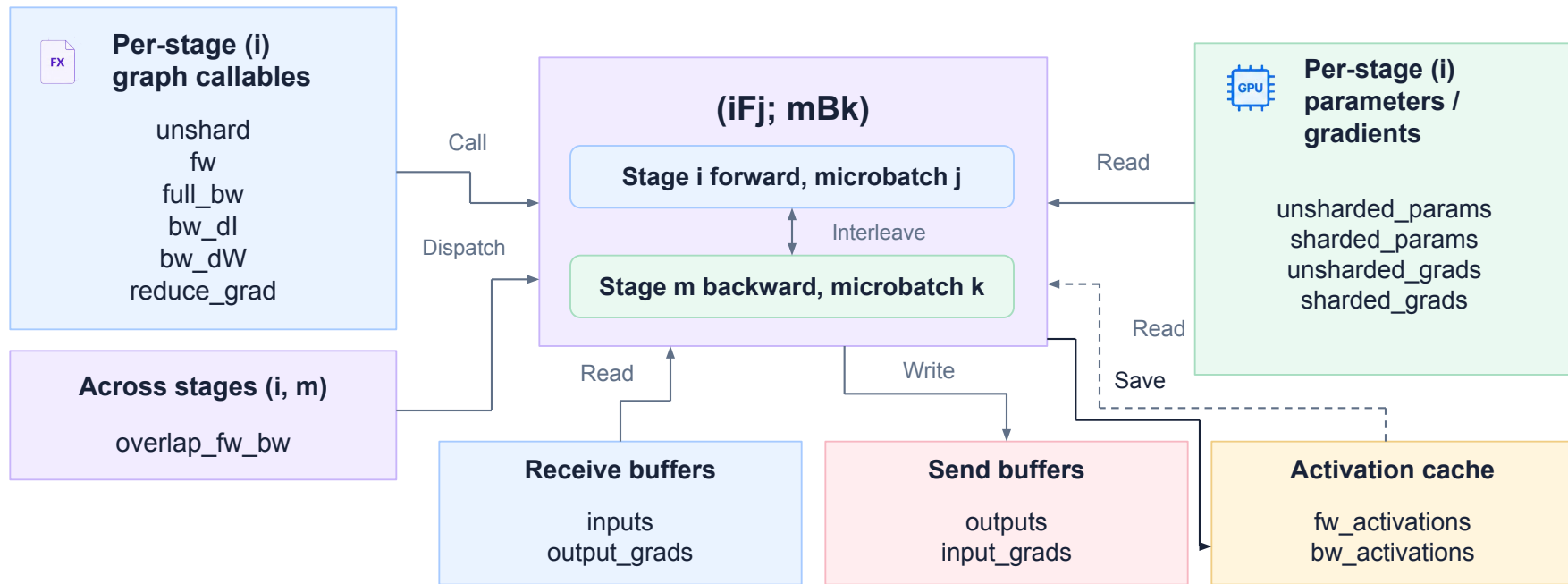
fw_activations
bw_activations



Per-stage (i) parameters / gradients

unsharded_params
sharded_params
unsharded_grads
sharded_grads

Graph-based PP Runtime: Overlap Execution



Performance at Scale

DeepSeek-V3 671B + MTP1 | 256 GB300 GPUs

Shared: EP64, SeqLen 4096, GBS 15360, MXFP8, DistMoE, CudaGraph, No AC, Forced Balance

SPMD: FSDP 256, MBS 1, Gradient Accumulation step 60

PP: PP2, VPP8, FSDP128, MBS 1, 120 microbatches

	Optimized Eager (tokens/s/GPU)	Graph Trainer (tokens/s/GPU)	Gain %
SPMD	5,591.63	5,656.79	+1.16%
PP2/VPP8	5,495.09	5,502.46	+0.13%

Compared with Megatron: 5,665.00 TPS/GPU (EP32, HybridEP page stashing)

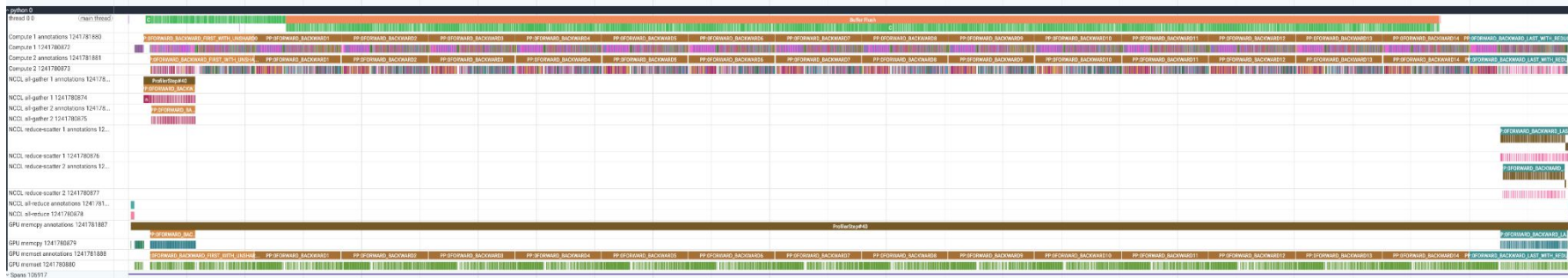
Bitwise-identical with eager across all 60 steps in both configurations.

Training Artifacts & Reproduction Instructions



Repro instruction and training artifacts can be found [here](#)

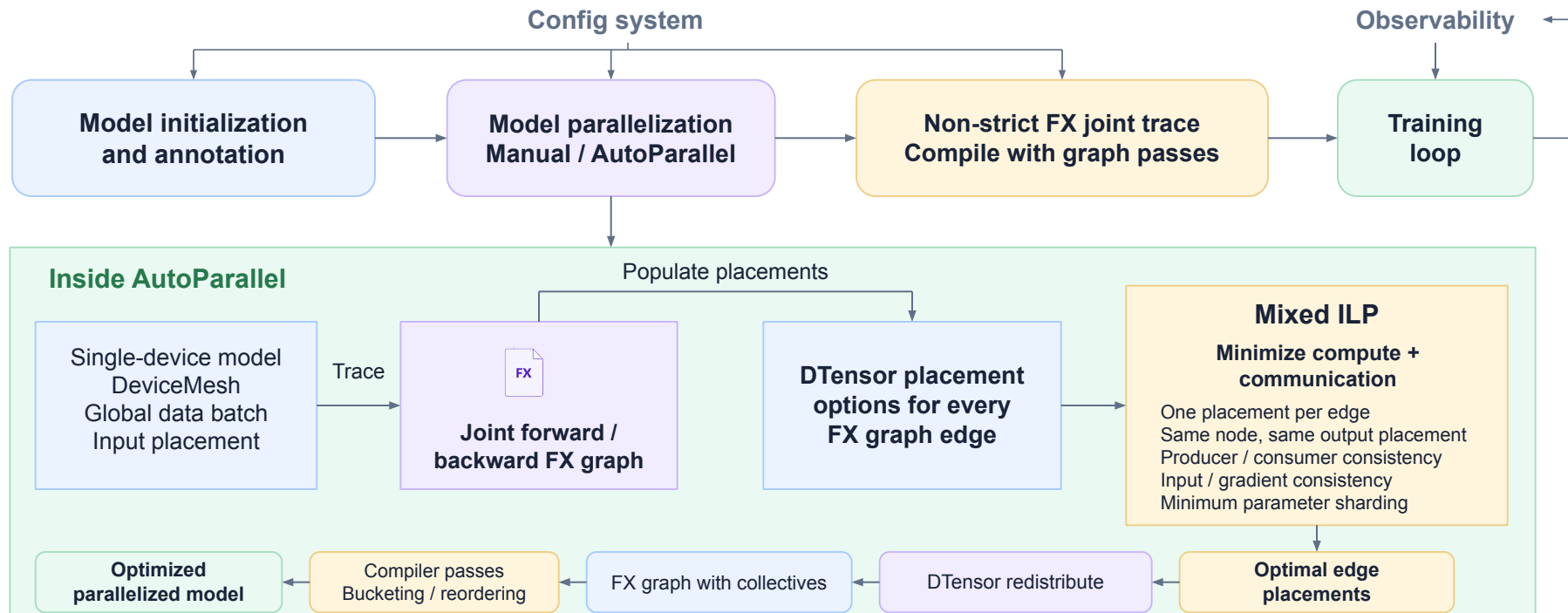
- Profiling trace
- Training logs
- Loss curves



The background is a vibrant, abstract composition of overlapping, wavy lines in various shades of red, pink, and orange. A prominent dark grey curved shape sweeps across the right side of the frame. The overall effect is dynamic and modern.

Emerging Directions

AutoParallel: System Design



DTensor and DeviceMesh

Core abstractions for specifying and executing an SPMD strategy

Shard (S)

Shard a tensor dimension across a device-mesh dimension.

Replicate (R)

Replicate a tensor across a device-mesh dimension.

Partial (P)

A tensor pending reduction across a device-mesh dimension.

Data to distribute: $[[0, 1], [2, 3]]$

0	1
2	3

$[S(0), S(1)]$

$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$	$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$
$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$	$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$

$[R(), R()]$

$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$	$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$
$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$	$\begin{bmatrix} 0, 1 \\ 2, 3 \end{bmatrix}$

$[S(0), R()]$

$\begin{bmatrix} 0, 1 \\ 0, 0 \end{bmatrix}$	$\begin{bmatrix} 0, 1 \\ 2, 0 \end{bmatrix}$
$\begin{bmatrix} 0, 0 \\ 0, 3 \end{bmatrix}$	$\begin{bmatrix} 0, 0 \\ 0, 0 \end{bmatrix}$

$[P(), P()]$

Example matmul: $C = A @ B$

(R, R) gives R

A	B	C
---	---	---

(S(0), R) gives S(0)

A	B	C
---	---	---

(R, S(1)) gives S(1)

A	B	C
---	---	---

(S(1), S(0)) gives P()

A	B	C
---	---	---

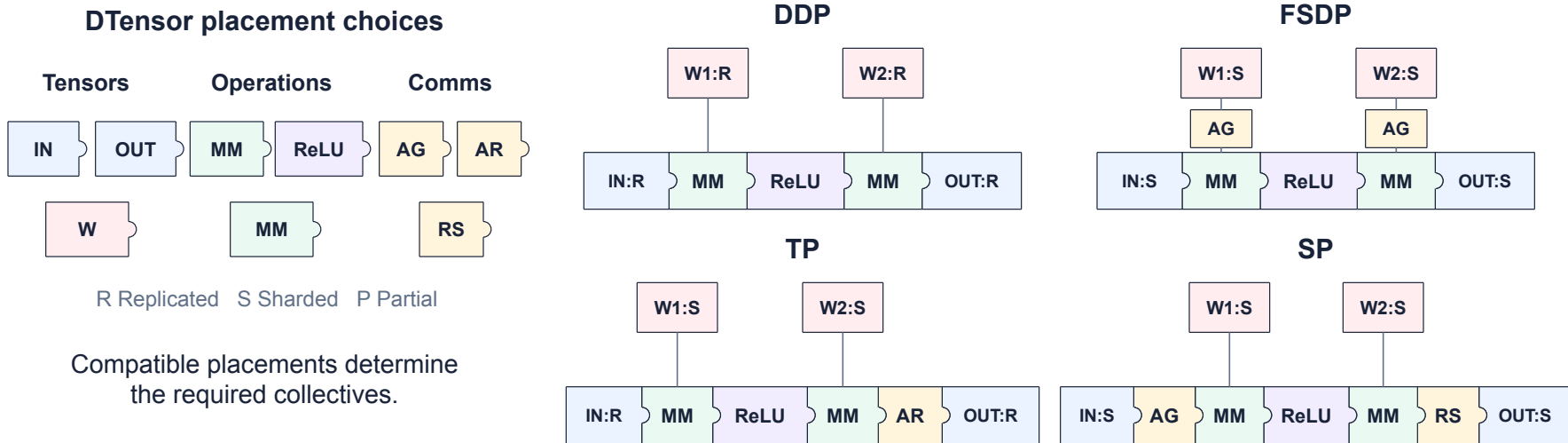
DeviceMesh

0	1
2	3

Global device topology
2-D mesh $[[0, 1], [2, 3]]$

AutoParallel: Optimizing Model Sharding

DDP, FSDP, TP and SP combine DTensor placements and collectives.



AutoParallel minimizes cost while satisfying global placement constraints.

Extending Across Hardware Backends

TorchTitan TPU Training is prototyping on top of GraphTrainer.

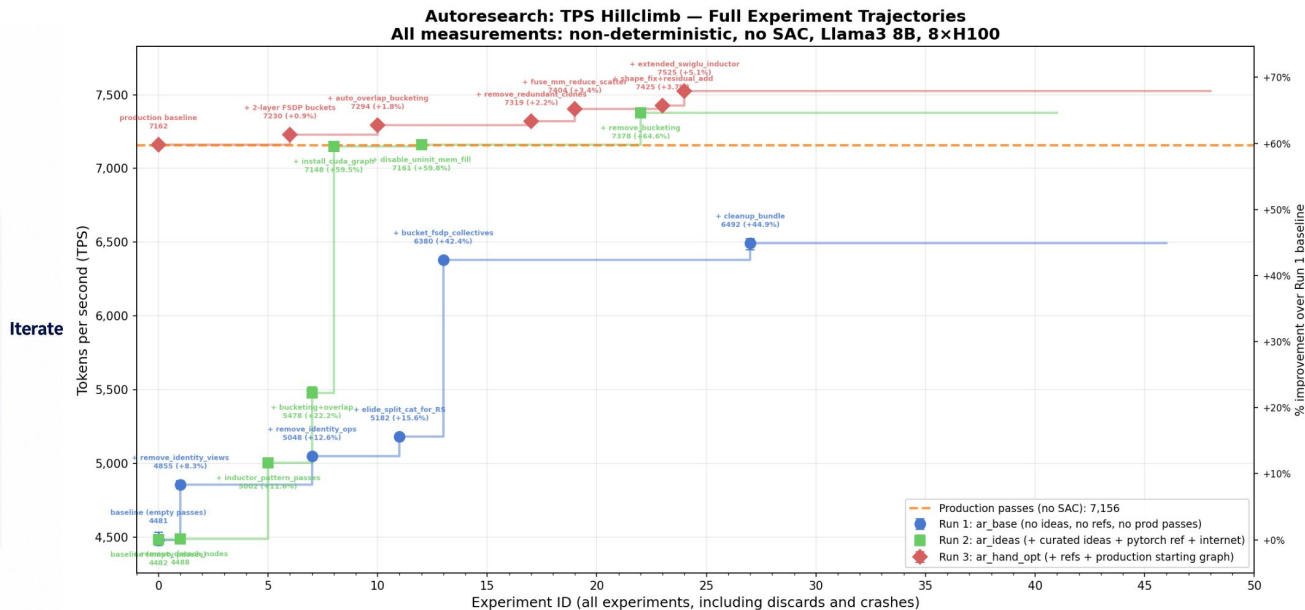
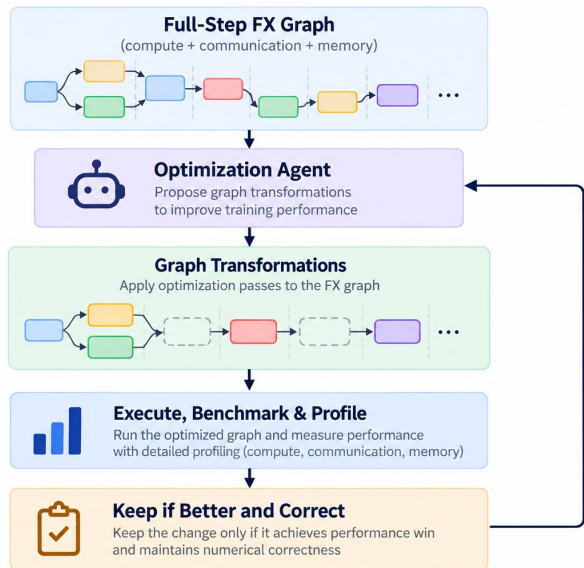
- Reached 29%+ MFU on Qwen3-235B
- Optimizer can join the full-step graph
- more details at TorchTPU talk

Hardware Ecosystem

- Thanks contributions from NVIDIA, AMD, and Intel

Graph as Substrate for Agentic Hillclimb

- Agent optimize the FX graph, not modeling or infra code
- Structured transformations with verifiable graph invariants



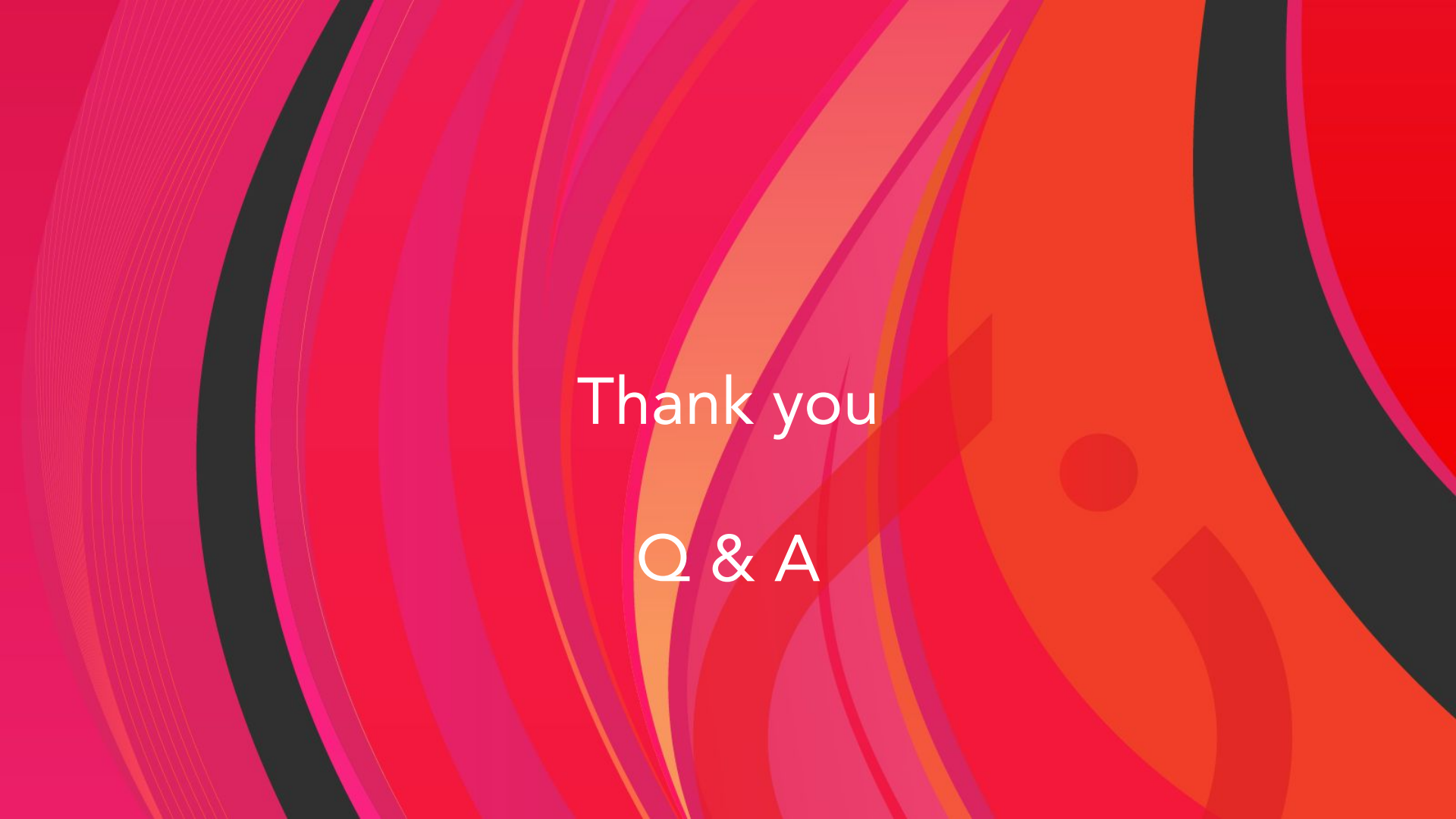
Recap

GraphTrainer is a toolkit for graph-based LLM training

- Minimal FX Tracer provides a unified training graph
- Optimizations as composable graph passes
- Supports FSDP, EP, TP, CP, PP
- Performant at scale
- Supports multiple hardware backends
- Programmable substrate for agentic optimization



Join our [slack user group](#)



Thank you

Q & A