



CONFERENCE

—NORTH AMERICA 2026—

Ray All the Way Down: A Heterogeneous, Elastic PyTorch Training Stack at LinkedIn

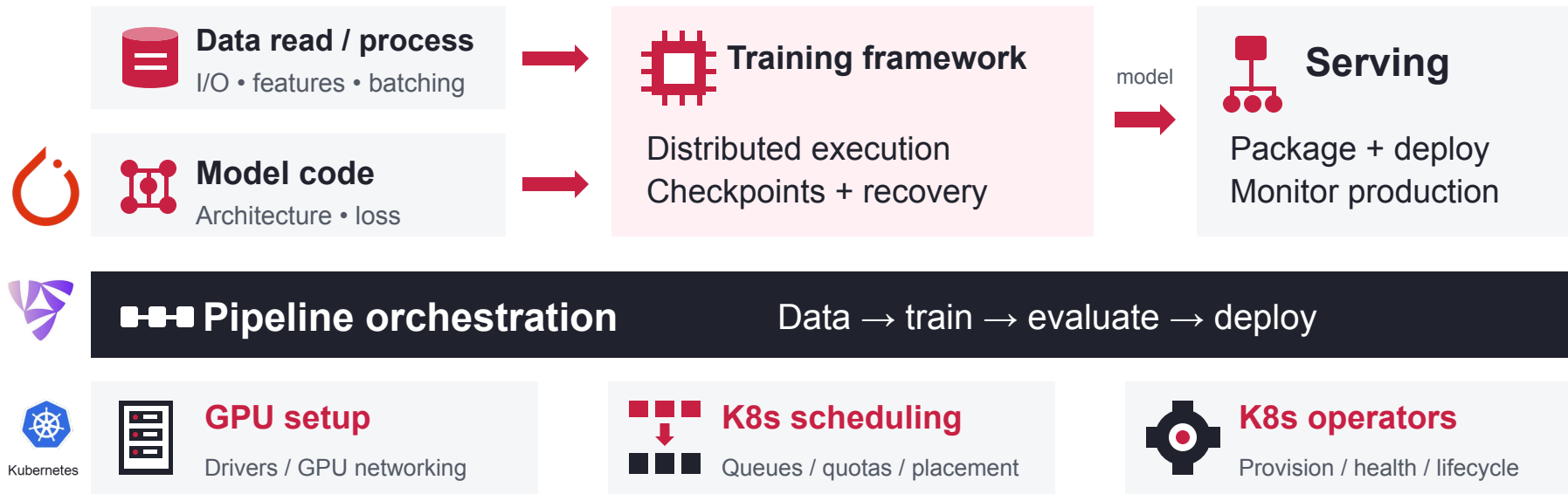
Tommy Li - Staff Software Engineer, LinkedIn

Tao Huang - Senior Software Engineer, LinkedIn



LinkedIn's challenge: the whole on-prem stack

We operate the on-prem infrastructure and connect the full ML lifecycle.



A faster training loop is not enough. Efficiency and reliability must span every layer.

Ray, end to end

A heterogeneous, resilient PyTorch training stack

- Scale data and compute independently.
- Make recovery part of the architecture.

One runtime. Clear responsibilities.

Ray Core

Data pipelines



Ray Train

Worker lifecycle



PyTorch

Distributed compute



One job, one coordinated training stack



Keep data on CPUs. Keep GPUs training. Let Ray coordinate the job.



Platform lifecycle

Kubernetes + KubeRay

Images • placement • credentials • monitoring

Submit once



Config + model
+ runtime image



One heterogeneous Ray cluster

Ray Core data actors



Read + preprocess

ready batches



Ray Train coordinator



PyTorch GPU workers

async save



Durable state
+ artifacts

Native PyTorch path shown; framework adapters can supply a different training loop.

DDP • FSDP2 • HSDP

Replicate • shard • shard within / replicate across groups

Resume complete checkpoints only

Publish artifacts, then tear down the cluster.

One framework, many training recipes



Keep the training choice. Reuse the platform integration.

SFT



Learn from labeled examples

Distillation



Teacher → student via custom loss

Custom experiments



Swap architecture, loss or loop

PyTorch / custom

Lightning

Hugging Face

Megatron

Shared Ray-based training framework

Worker launch • CPU/GPU resources • run metadata • artifact paths

Less platform glue

Reuse launch, images and observability

Backend-specific data and checkpoint integration remain required; elastic resizing is not automatic.

Faster experiments

Change the recipe, not the cluster setup

Shared operations

Common run metadata and artifact paths

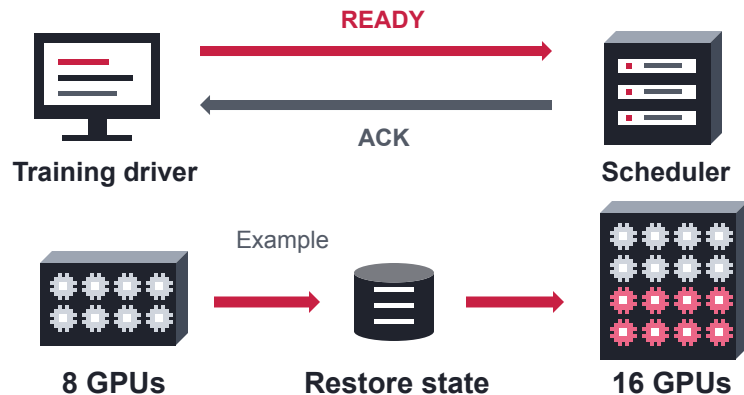
Elastic training and RL on Ray

LinkedIn shares the orchestration platform; specialized engines do the compute.

Elastic training



Complete checkpoint + shared-file handshake



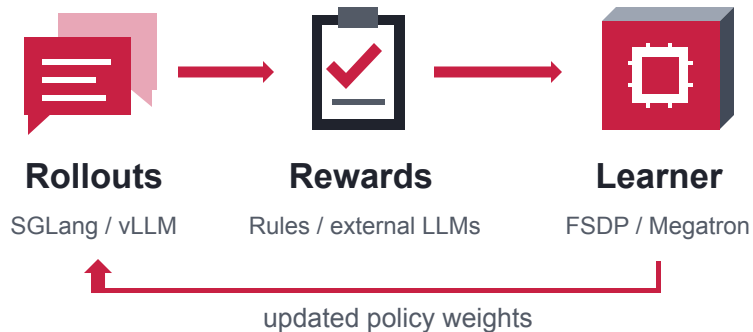
ACK: stop + resize; new ranks restore + reshard.

No ACK: continue. No hot-added ranks.

RL with verl + Ray Core



Actors • flexible CPU/GPU pools • platform reuse



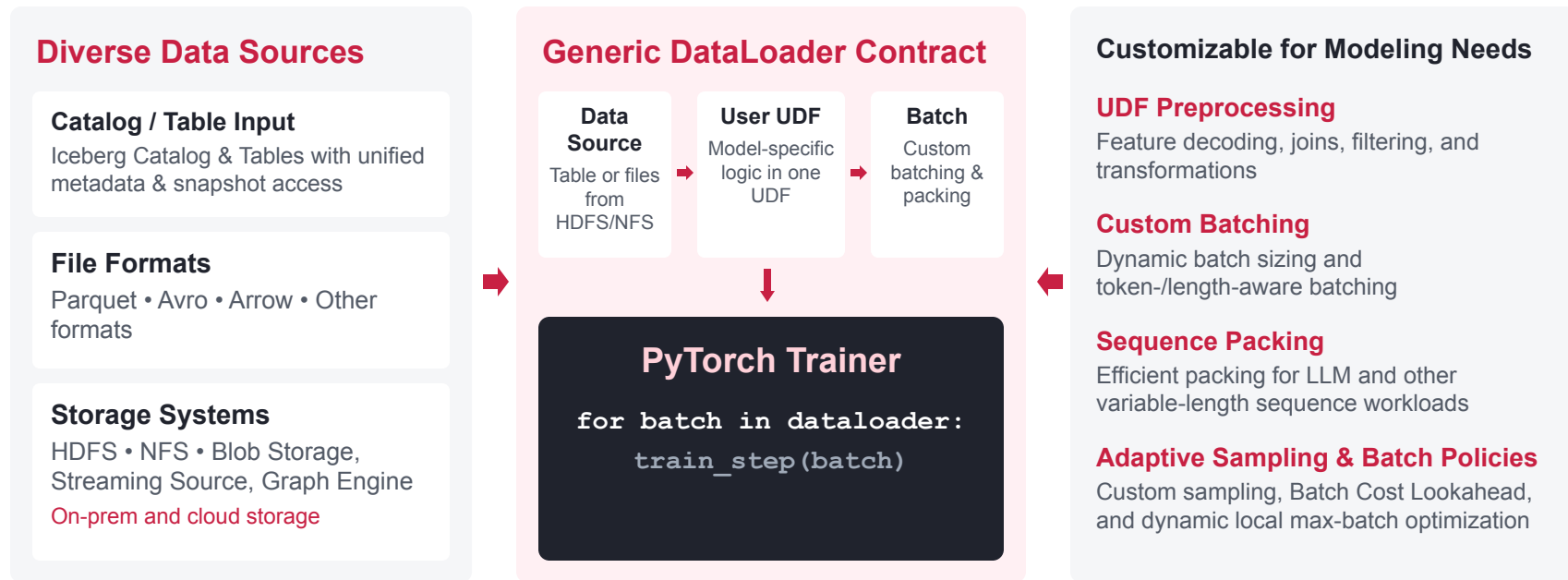
~20× faster weight updates (verl + SGLang); not end-to-end.

PPO / GRPO + multi-turn agents; async RL is experimental.

Full-state recovery still needs checkpoints.

A Generic DataLoader Abstraction for Diverse PyTorch Workloads

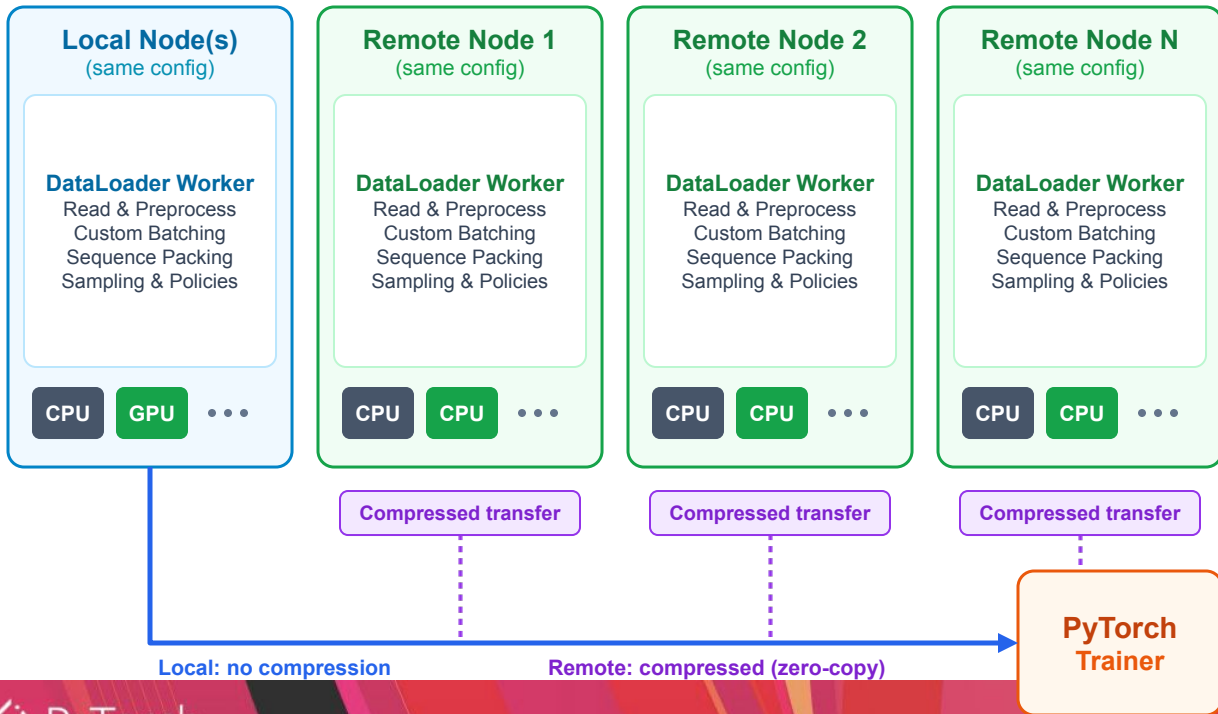
One contract, many data sources, and fully customizable data processing for modeling needs



Linear, Cross-Node Scaling with Ray Core

Same DataLoader contract and configuration across local and remote nodes

Same DataLoader config (UDF, batching, packing, sampling) • Scale out to remote nodes



Performance & Impact

2–3×

vs. out-of-box Ray Data

Higher data loading throughput with better customization

\$1M+

GPU hours saved

More efficient data pipeline at scale (per workload average)

5× p50

speedup vs. torch DataLoader

Across 100+ real production use cases

Control plane flexibility + distributed data-plane efficiency

Acknowledgements & Poster Presentation

PyTorch Framework Integration

Jaideep Ray, Jonathan Hung, Chen Zhu, and LinkedIn's Training Framework & Dragonknight teams.

Kubernetes Integration & Elastic Scheduling

Santosh Jha, Zhuo Zhi, and LinkedIn's Kingkong & OpenConnect teams.

Open-Source Foundations

Thank you to the PyTorch and Ray communities for the foundational frameworks that make scalable model training efficient.

POSTER PRESENTATION

Batch Cost Lookahead: Coordination-Free Workload Regularization for Packed Variable-Length Training

Join us at our poster session to learn more!

WHEN & WHERE

Oct 20 • 6:00 PM – 7:30 PM

Community Expo

Thank You!