PyTorch CONFERENCE

—NORTH AMERICA 2026—

Building a Post-Training Platform for Teams That Don't Own the Training Loop

Gaurav Arora Tech Lead, AI Training Platform, Pinterest
Shunyao Li Senior Software Engineer, AI Training Platform, Pinterest
Eric Wang Senior Software Engineer, AI Training Platform, Pinterest

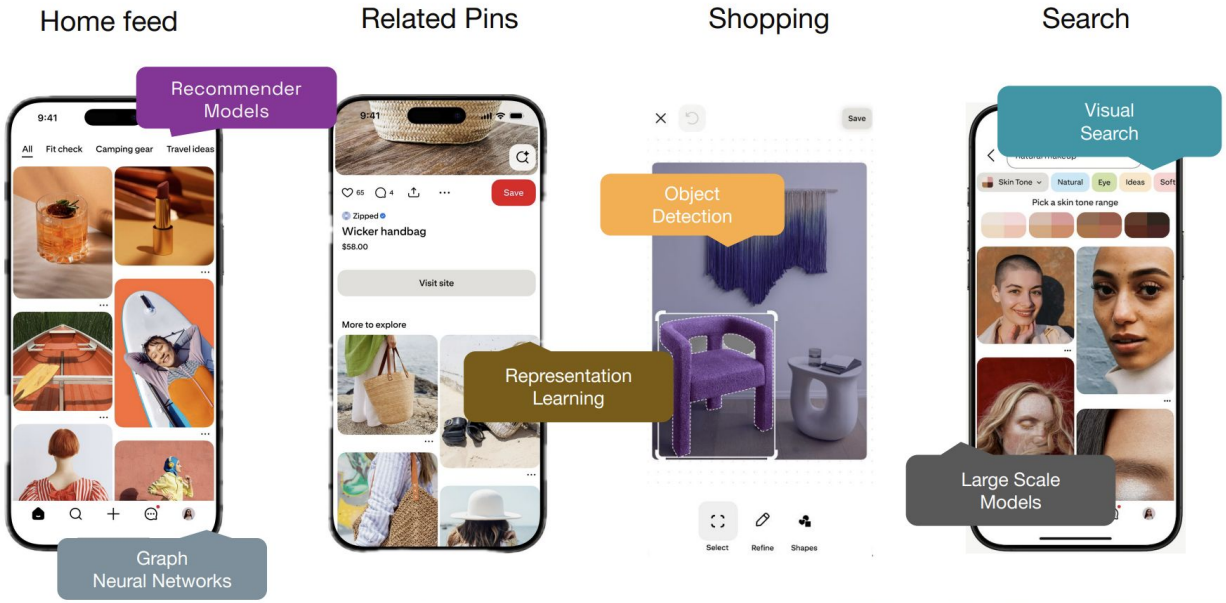
AGENDA

What we'll cover

01	**Why a post-training platform**	What MLEnv taught us, and what post-training broke
02	**PTEnv architecture**	Own the lifecycle, delegate the inner loop to OSS frameworks
03	**Offline journey**	PinCLIP signals · efficiency
04	**RL journey: a tool-using agent**	Environments · bring-your-own-harness · weight sync
05	**Lessons**	what's proven today · takeaways

ML at Pinterest

Recommendation and content-understanding models power every surface.



Now also: post-trained foundation models

Product teams take open foundation models and post-train them for their surface.

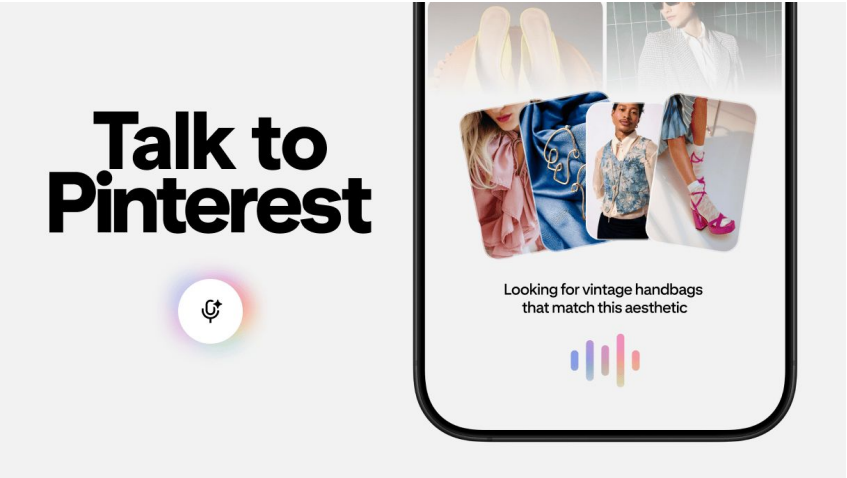
What we post-train

open VLMs, adapted per product

- Navigator
- Annotation judge
- PinCLIP reranker
- Trust & Safety VLM
- Product tagging
- 1P CIM

SFT · DPO · GRPO

Qwen3-VL · Qwen3.5 / 3.6 · Gemma 4 — from 4B dense to 235B MoE



Pinterest Assistant, powered by Navigator

same company, a new kind of training job: the loop lives in someone else's framework

Our ML training platform

MLEnv runs classic training; PTEnv is emerging alongside it for post-training.

WORKLOADS

- Batch inference
- HPT experiments
- Last-mile data processing
- ML training jobs
- **Post-training · SFT · DPO · GRPO**

ML RUNTIME

PyTorch MLEnv SDK

- Data pipeline SDK (Ray)
- Batch inference SDK
- DDP / FSDP / DMP training loop
- Training launcher (local → TCP)

PTEnv · post-training (NEW)

- recipes · contracts · eval / export
- *training loop: OSS frameworks*

W&B
experiment tracking / model registry

Ray · distributed compute engine

COMPUTE INFRA

Training Compute Platform (TCP)

- ML compute control plane
- ML compute data plane
- PinCompute EKS

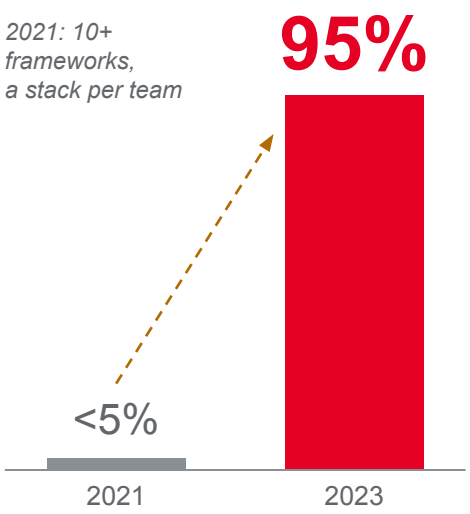
SUBSTRATE

AWS GPU instances · H200 (P5en) · A100 (p4d / p4de) · B200 (P6) · Ultracluster · EFA / RDMA

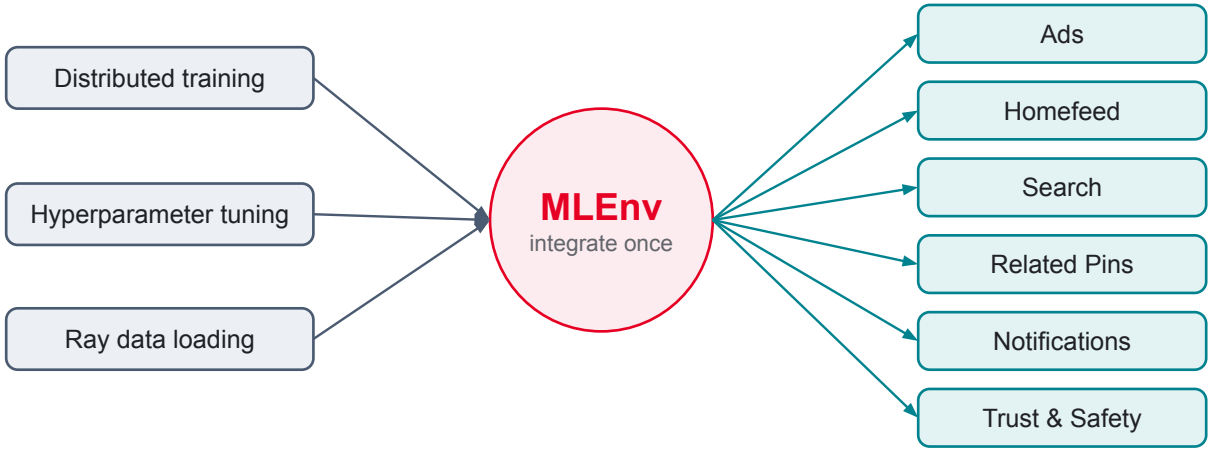
What MLEnv taught us: integrate once

Standardization made infrastructure a one-time integration that every team inherits.

share of Pinterest ML jobs on MLEnv



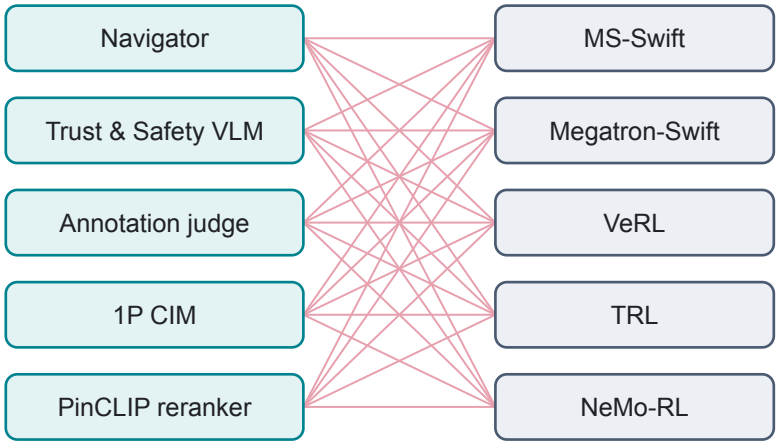
Source: "MLEnv: Standardizing ML at Pinterest Under One ML Engine", Pinterest Engineering Blog, Sep 2023. Share of ML jobs, not teams.



integrate infrastructure once · let capabilities spread to every team
and teams keep native control of data loading, model architecture and the loop

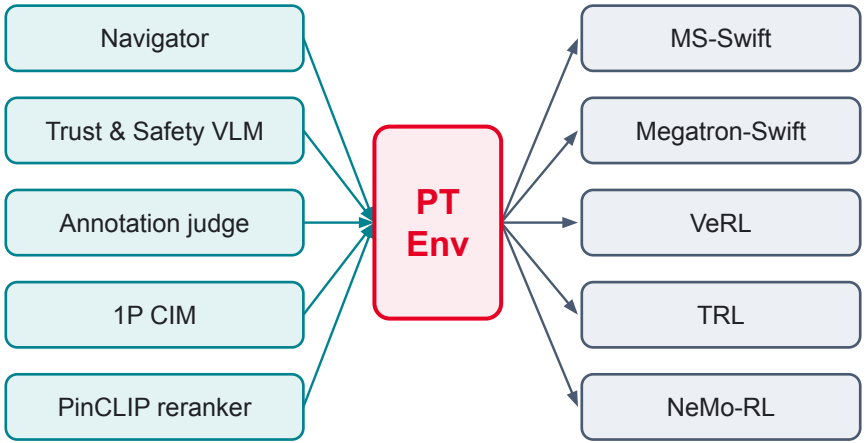
Post-training brought integration fragmentation back

No single framework fits every workload, and they ship monthly. So every team re-solved data, checkpoints, export, eval.



up to N × M = 25 bespoke integrations

each team re-solves data, checkpoints, export, eval



N + M = 10 integrations

recipes · contracts · adapters

PyTorch is the stable bet. The frameworks on top of it aren't.

PTEnv owns the lifecycle. The framework owns the loop.

One contract for every backend, with the common work built once as shared services.

```
# the whole contract · base.py
def run(self):
    try:
        self.setup()
        self.train_result = self.train()
        self.evaluate()
        self.export()
    finally:
        self.cleanup()
```

`setup()`	model resolution · data staging · tracking	PTEnv
`train()`	Megatron-Swift · MS-Swift · VeRL	FRAMEWORK
`evaluate()`	generation eval suites + scoring	PTEnv
`export()`	validate + publish the HF model	PTEnv
`cleanup()`	always, even when a stage fails	PTEnv

Built once
shared services

- model resolution
- generation eval
- validate + publish

Megatron-Swift and MS-Swift
call the same code

Teams pick the model. PTEnv picks a tested backend.

Each model has a tested default backend. Teams can override it; an unsupported choice is rejected, never silently swapped.

```
# the team writes
model: Qwen3-VL-235B-A22B
task:
  objective: sft
tuning:
  method: lora
data:
  train: s3://…/train.jsonl
backend: default  # or ms_swift
```

PTEnv resolves

tested default for this model
Megatron-Swift · EP=8 · FP8 · 1 × 8 B200

override is qualified
→ keep it

override not qualified
→ reject, with the reason

TESTED DEFAULTS TODAY

Qwen3-VL-235B-A22B · Qwen3.5-397B-A17B · Qwen3.8-Flash-Next	**Megatron-Swift**	expert / tensor / pipeline parallel · FP8
Qwen3.6-35B-A3B · Qwen3-0.6B · Gemma4-26B-A4B	**MS-Swift**	HF trainer · LoRA or full · DDP
Qwen3-4B (ads next-SID SFT) · Qwen3-VL-30B-A3B (PinCLIP DPO)	**TRL**	HF SFT / DPO trainers · LoRA or full
GRPO · Qwen3-VL-8B / 30B-A3B · annotation judge, agent RL	**VeRL**	trainer on Ray · FSDP2 + vLLM rollout

Start a fine-tuning job

Example: LoRA SFT of Qwen3.8-Flash-Next. The spec stays small; the recipe pins everything that was tested together.

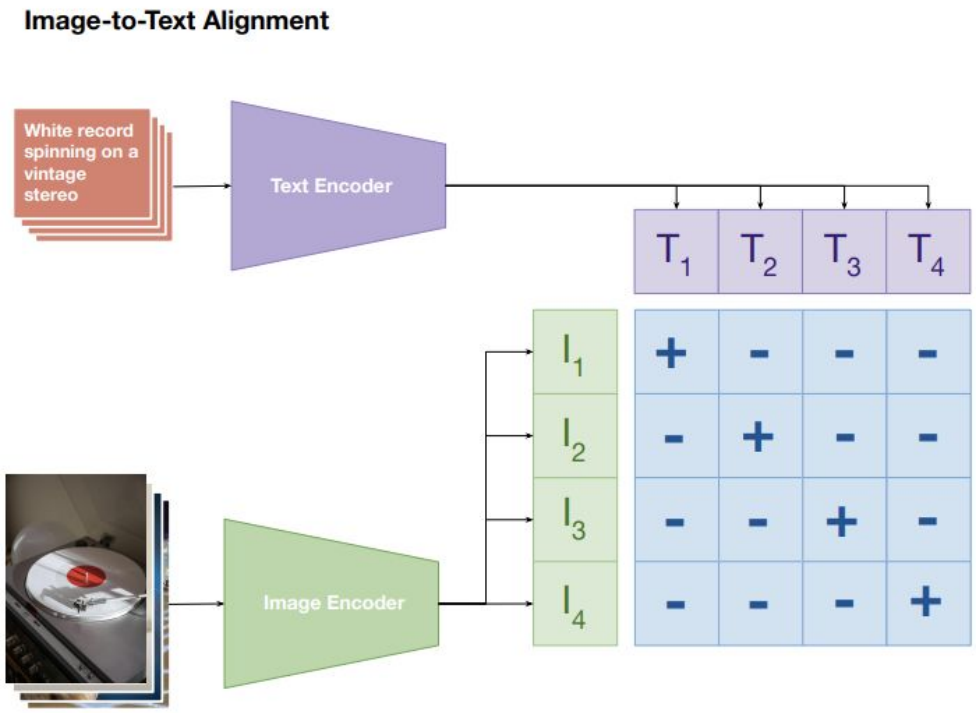
```python
from trainer.ppytorch.post_training.recipes import sft
from trainer.ppytorch.post_training.spec import (
    FinetuneSpec, LoRASpec)

SFT_JOB = sft.qwen3_8_flash_next(
    FinetuneSpec(
        train="s3://my-bucket/train.jsonl",
        output="s3://my-bucket/flash-next-sft/",
        max_epochs=1,
        tuning=LoRASpec(rank=8, alpha=32,
            target_modules="in_proj out_proj "
                           "linear_proj linear_qkv"),
    )
)
```

Who owns each step of this job

PTEnv	plan	spec + recipe checked on CPU; all-linear LoRA rejected up front
	stage	training data and Hugging Face weights staged, before any GPU
backend Megatron-Swift	prepare	1 node · 8×B200 · TP2 / PP2 / EP4; HF weights loaded across ranks
	train	native loop · FlashAttention 2 · 2048 tokens, global batch 8 · n-gram embeddings on CPU
	package	LoRA adapter exported as a loadable model
PTEnv	evaluate	eval suites on the exported package
	publish	Exported model to S3

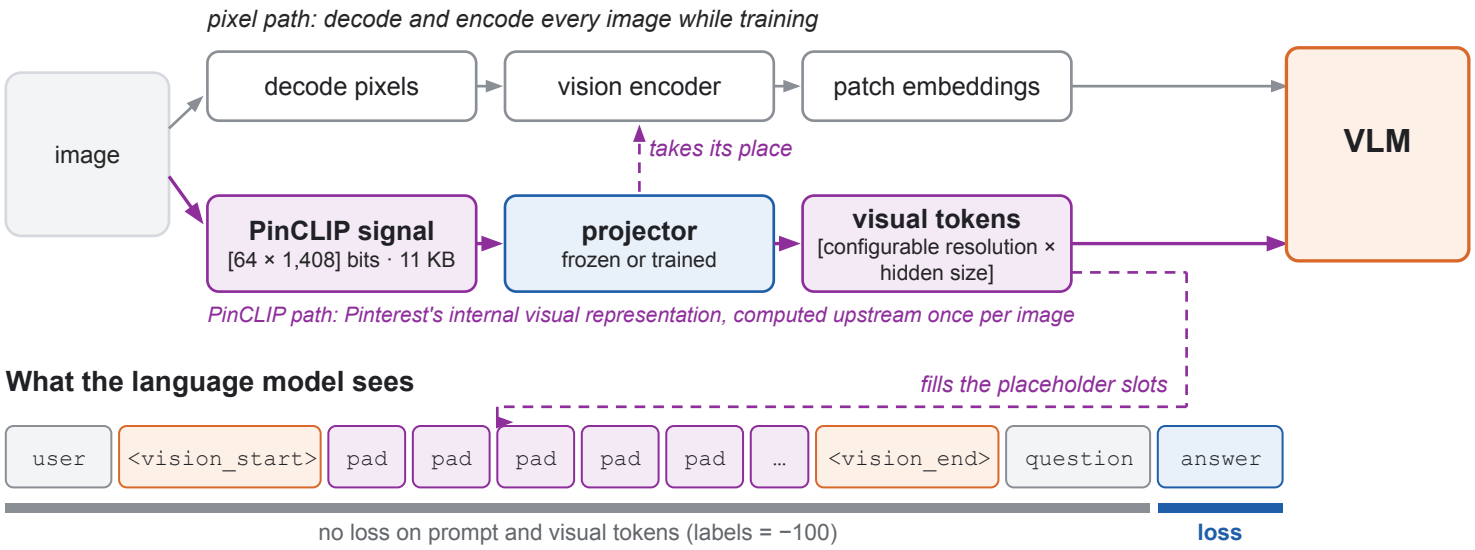
PinCLIP: Pinterest's own visual signal



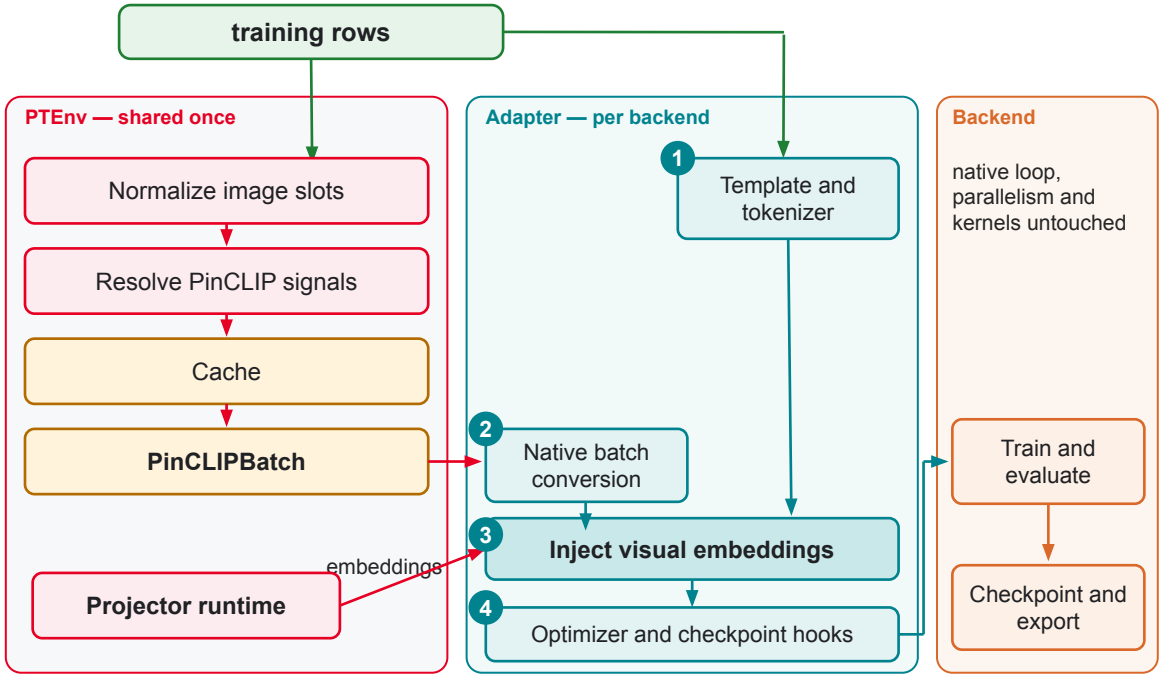
- Foundational Multimodal Representation @Pinterest
- Multimodal embedding model that operates on images and text inputs.
- Designed to be used for retrieval with image and text queries.

PinCLIP Signals in VLM

Replace pixel decoding and the vision encoder with a precomputed PinCLIP signal.



Adapt PinCLIP signals in PTEnv



EXAMPLE · THE MS-SWIFT ADAPTER

a plugin on Swift's own extension points

1. **PinCLIP-aware Qwen template**
   reads each image's packed signal; expands its placeholder to n visual tokens and masks them out of the loss
2. **Collator**
   stacks per-row signals into the batch tensors the model expects
3. **Model loader and forward**
   wraps Qwen with the shared projector at load; its output replaces the placeholders, pixel and PinCLIP images keep their order
4. **Projector as Swift's “aligner”**
   Swift's own freeze and learning-rate options apply; saved with checkpoints

Out of the box isn't tuned

Why we spent time on SFT efficiency, and what it bought us.

THE ASSUMPTION

"Popular framework + popular open model = already tuned"

Defaults are chosen to be safe and general, not fast for your workload.

→

WHAT WE FOUND

≈ 2×

SFT throughput

same 8×B200 node · same models · no model-specific code

→

WHY IT MATTERS

data mix → train → eval ↺

Post-training is a loop: 2× means twice the iterations per day, or half the GPU cost.

Found once, pinned in the recipe: every team gets it by default.

SFT Training Efficiency

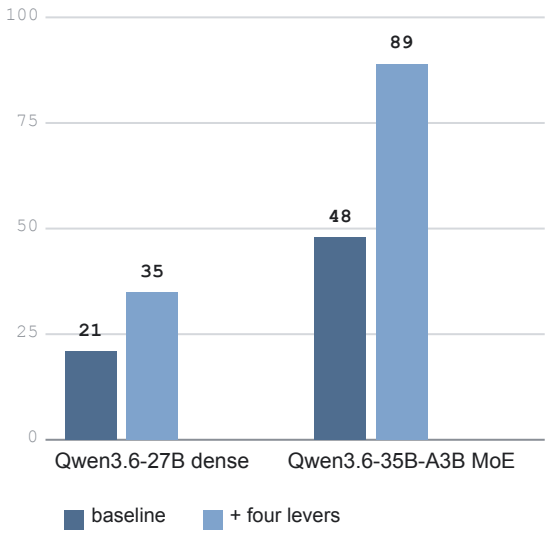
Model-agnostic levers gave 85% win, mostly from padding-free.

Optimization #1

Four model-agnostic levers

flash attention	instead of sdpa
lighter checkpointing	non-reentrant recompute
padding-free	no compute spent on padding
bigger batches	the saved memory allows

MEASURED · EXAMPLES / S · 8×B200



SFT Training Efficiency

A profile then showed memory-bound, and Liger's fused kernels improve it.

Optimization #2

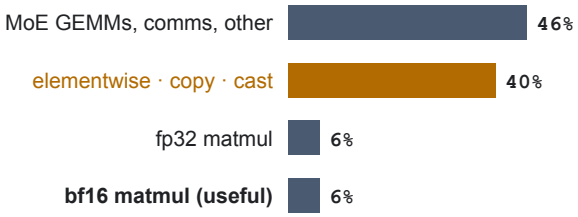
Fuse with Liger Kernel

hidden → logits bf16 → fp32 copy → loss

hidden → **one fused, chunked kernel** → loss

logits never hit memory · RMSNorm fused too

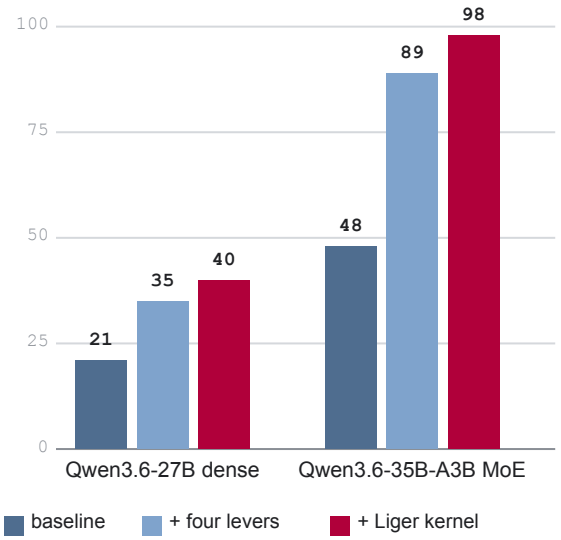
GPU KERNEL TIME · QWEN3.6-35B-A3B



≈ 40% of time moves data: reads, writes and casts, little math.

The hidden cost is the loss: it builds the logits over the whole vocabulary, the largest tensor in the step, then copies it to fp32.

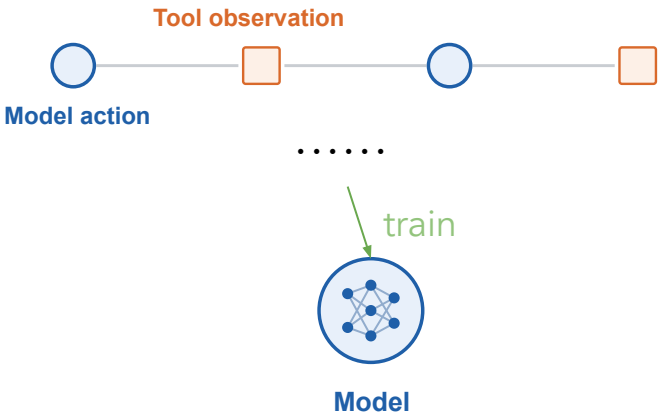
MEASURED · EXAMPLES / S · 8×B200



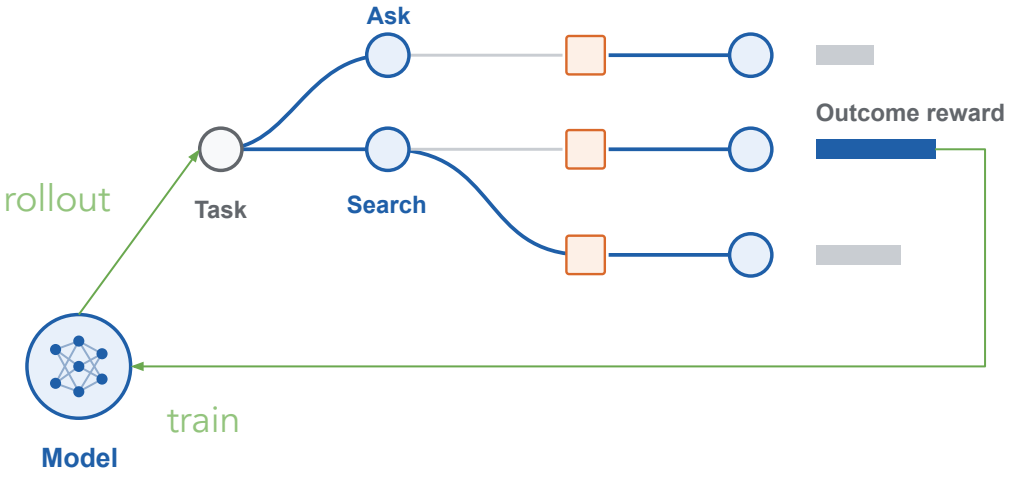
RL journey: Why reinforcement learning matters

SFT learns from demonstrations. RL learns from the outcomes of new attempts.

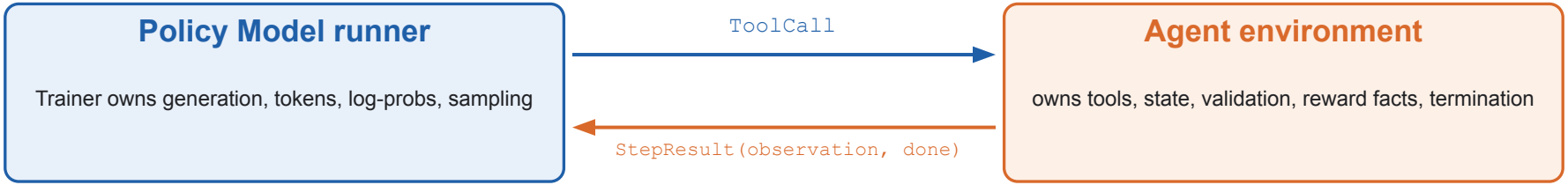
SFT: imitate demonstrated static paths



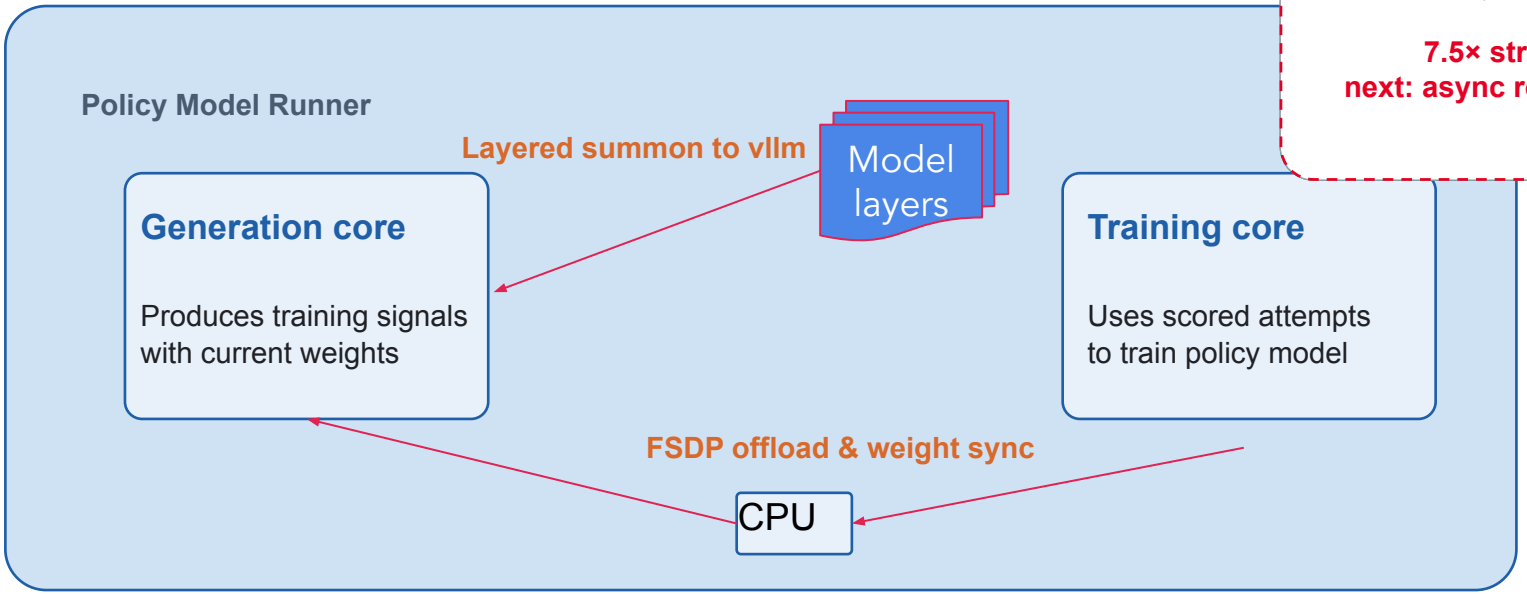
RL: explore through iterations



Environment Policy Isolation



Efficiency Optimization: Bottlenecks



③ GENERATION · NEXT

Once sync is bounded, generation is 76% of the step

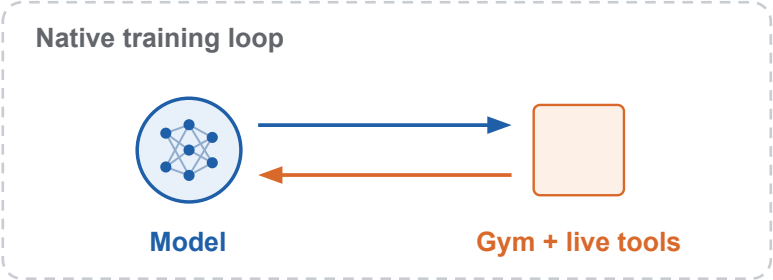
30B VLM, 16 samples per task

7.5× straggler gap
next: async rollout, est. −18%

Trainer-native vs bring-your-own-harness

Trainer-native

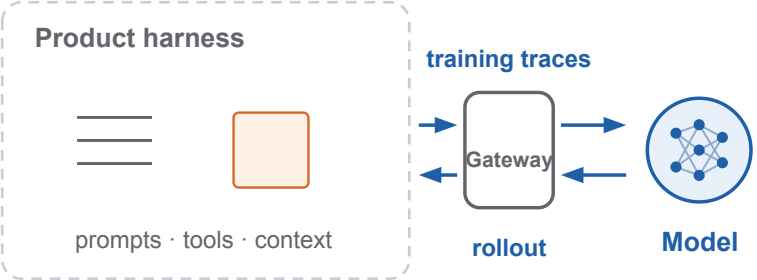
Trainer framework runs the full loop



Use the shared loop when actions and observations fit it cleanly.

Bring your own harness (BYOH)

Product keeps its existing loop



Reuse production orchestration when rebuilding it would cause drift.

What is demonstrated today

Proven

- Megatron-Swift SFT, incl. 235B MoE FP8 on one 8×B200 node
- Navigator SFT at ~2× throughput
- colocated RL: 2.6× faster step after the sync fix

Experimental

- BYOH via the Rollout Gateway
- PinCLIP on Navigator Swift, the reference adapter

Next

- TRL backend and recipes
- PinCLIP adapters for other backends
- Asynchronous RL

TAKEAWAYS

What other platform teams can take away

1

Own the life cycle, delegate the loop

Shared data, validation, tracking, eval and export, without forking frameworks that ship monthly.

2

One contract, many frameworks

Framework choice becomes a config line; execution and distributed strategy stay native.

3

Make correctness explicit

Exact sampled tokens, observation masks, one policy version per rollout, manifest-last checkpoints.

4

Measure, and define done

Profile before optimizing. Done means the exported model loads back and evaluates.

Acknowledgements

- AIP: Shunyao Li, Eric Wang, Dave Chen, Luyan Wu, Korhan Citlak, Gaurav Arora, AI Training Team
- ATG: Tianjian Huang, Raphael Poulain, Yash Upadhyay, Zhenyu Tan, David Xue, Joyce Yuan , Kofi Boakye, Logan Jeon
- Trust & Safety: Wenke Zhang, Prithwish Chakraborty
- Leadership: Qingxian Lai, Karthik Anantha Padmanabhan, Bo Liu

Thank you

Questions?

Gaurav Arora · Shunyao Li · Eric Wang
AI Training Platform, Pinterest