WAYVE

Why Logging Isn’t Enough: Making PyTorch Training Regressions Visible in Practice

Sahana Venkatesh, Wayve

April 8 2026

Agenda

April 8 2026

01
The Happy State: Normal Training Workflow

02
Why we care about regressions

03
How do regressions look in practice

04
What actually catches regressions early

05
Sharing some of our patterns

06
Future direction

Regressions

Regression : A change that makes the system worse than its previous known-good behavior

- Training gets slower
- Model quality is worse
- Expensive GPU and developer resources are wasted.

The Happy State : Normal Training Workflow

Healthy Training is both a model-quality story and a systems story

Model signals

- Training loss trends down in a stable way
- Validation metric improves or stays aligned
- Repeated runs are close enough that differences are interpretable

Runtime signals

- Step time and throughput are stable
- Memory stays within expected bounds
- No sudden stalls
- Failures are captured early before they become a problem

A run is “healthy” when it is converging, comparable to past work, and operationally stable enough that you trust the signal.

How do regressions look in practice

The failure mode is often visible just not automatically surfaced

Looks fine in train, worse in validation

Training curves can stay close while validation loss shifts meaningfully across runs.

Diverges later, not immediately

A run can appear normal for a few epochs and only break once a model bug compounds.

Training Stalls

Training could stall, and the logs might not clearly show when the stall actually began.

Performance regresses quietly

Final metrics may be similar while step time, memory use, or dataloader overhead get much worse.

Bisecting problems sources - High Level

Looks fine in train, worse in validation

Training curves can stay close while validation loss shifts meaningfully across runs.

Diverges later, not immediately

A run can appear normal for a few epochs and only break once a model bug compounds.

Trajectory checks Alert on shape changes: later divergence, earlier overfitting, or slower recovery.

Bisecting problems sources - High Level

Training Stalls

Training could stall, and the logs might not clearly show when the stall actually began.

Runtime budgets Profile step time, dataloader overhead, current global step, and memory so quality-neutral slowdowns are still caught.

Bisecting problems sources - High Level

Performance regresses quietly

Final metrics may be similar while step time, memory use, or dataloader overhead get much worse.

Baseline comparison Compare every new run against a known reference data, not just against itself.

Regression Check through two lenses

1. By timing

Online vs offline checks

ONLINE

Runs during training

- Heartbeat / liveness monitoring
- Global-step stall detection
- Immediate alerts when progress stops

Best for fast detection, but usually narrower in what it can explain.

OFFLINE

Runs after the training step

- End-of-run summaries
- Baseline comparisons
- Historical trend checks

Best for richer regression detection, but slower and less preventative.

Use both: online checks reduce detection time; offline checks catch deeper regressions.

2. By fidelity and cost

Three-tier checks

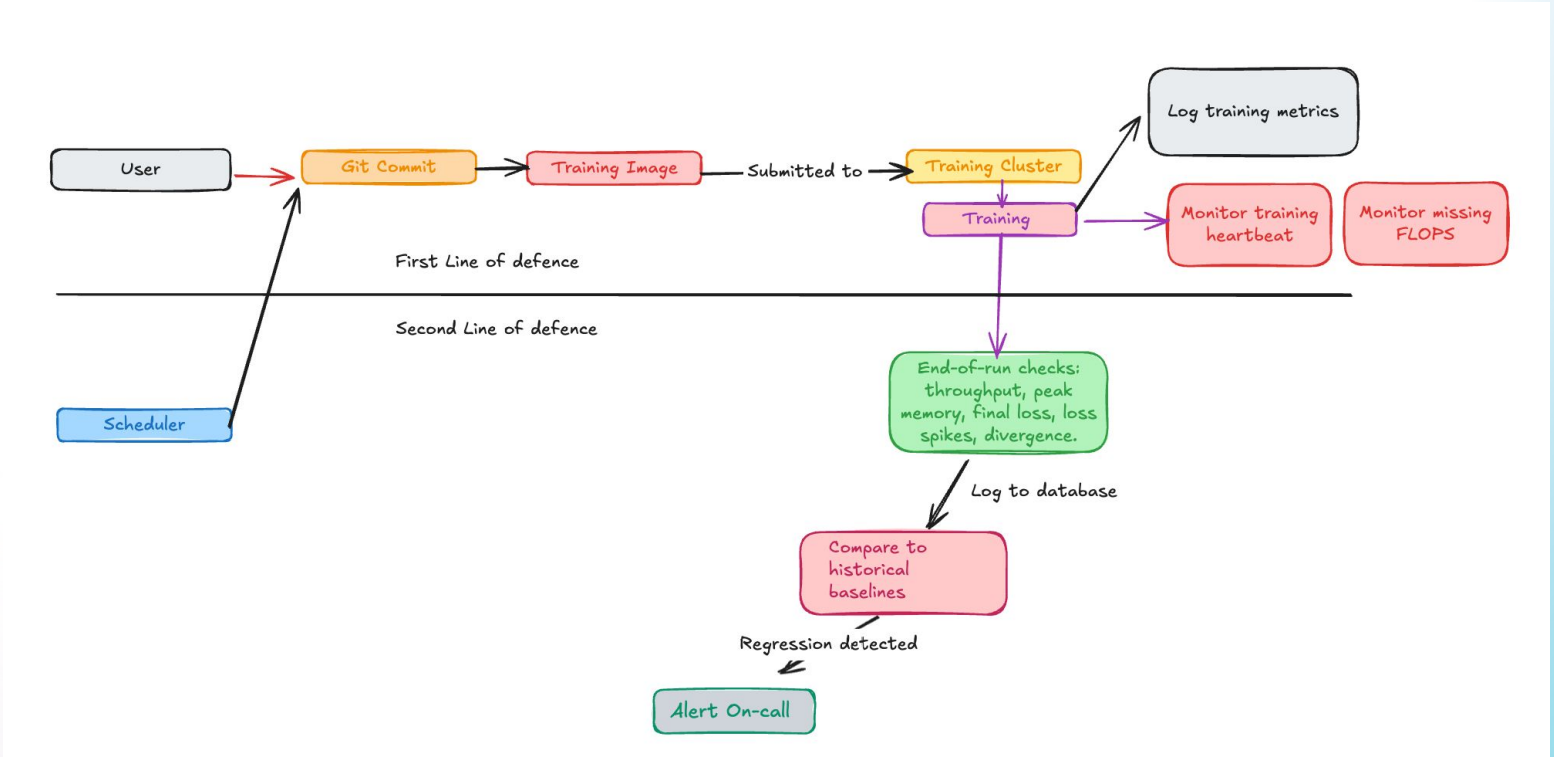
Unit tests
Fast, frequent; prevents obvious logic breaks from reaching main.
Least realistic

Integration tests
Checks component interactions after unit tests pass; slower but broader.
More expensive

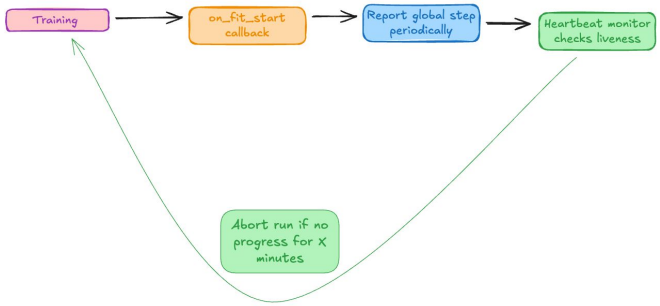
Federated tests
Closest to real-world behavior; catches training/runtime regressions near production.
Slowest; may not prevent merge

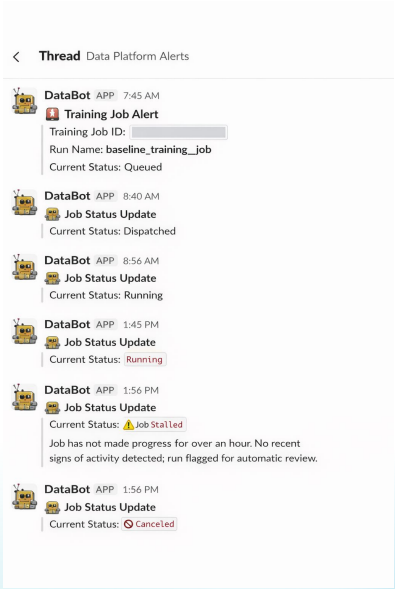
A practical system combines low-cost preventative tests with slower, high-fidelity regression checks.

Catching Regressions Early with Two-Tier Approach



Heartbeat-based stall detection



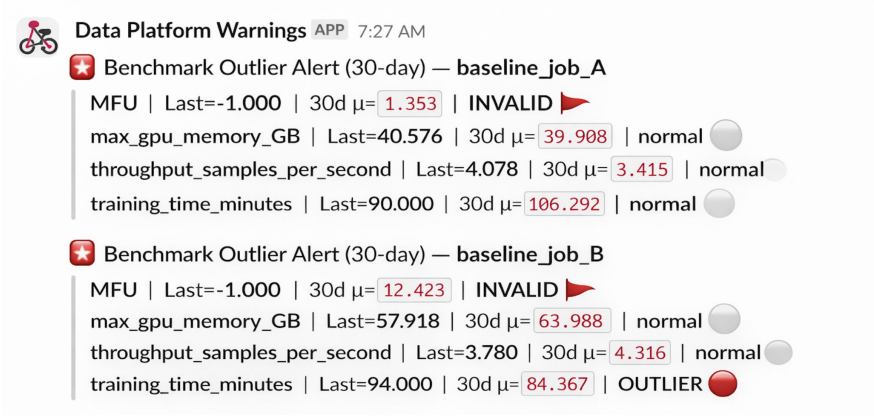


Heartbeat Monitoring

- Training job logs its latest global step in a database.
- If the global step has not changed over an hour, a cron job cancels the job.
- This saves developer time and minimizes wasted GPU resources.

Automatic stall detection surfaced through Slack before the failure became an expensive rerun

Using Historical trends for Regression detection



Regression Detection

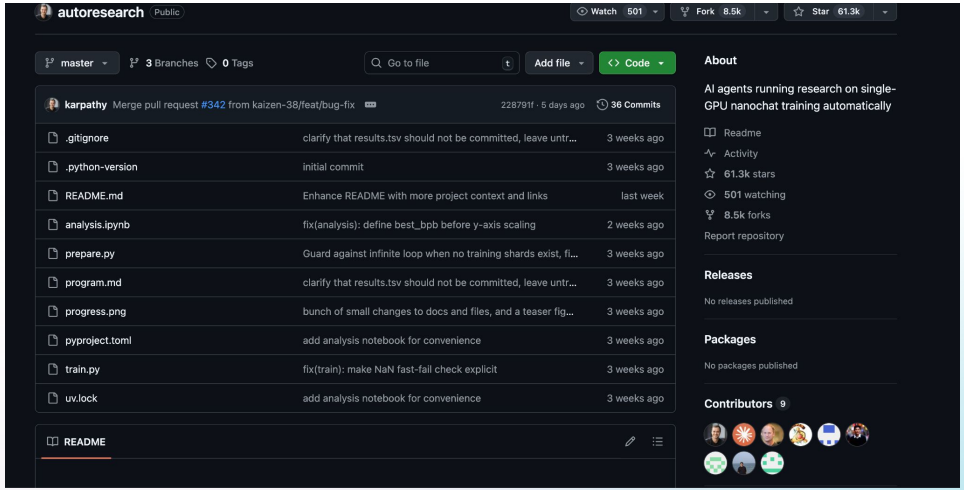
- A regularly scheduled trusted baseline acts as an early warning system for subtle but high-cost regressions.
- When these signals are tied to Git history, we can rapidly narrow down which commit introduced the regression.

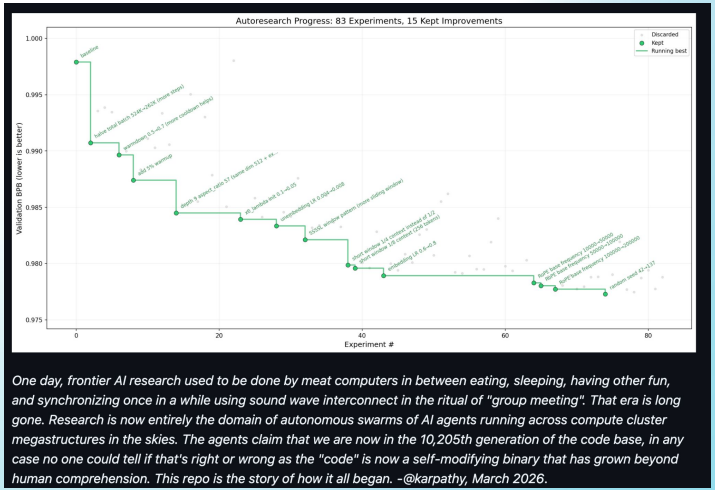
Automated outlier detection turned subtle performance regressions into actionable Slack alerts

Manual checks are future automation infrastructure

Turn logging into a set of explicit checks

- Agent-driven training loops are becoming real. autoresearch already demonstrates autonomous code changes plus train-evaluate-iterate cycles.
- But automation is only as good as its feedback signals.
- If we cannot reliably catch stalls and regressions, we cannot safely automate training.
- Building those checks now is a step toward consistently producing high-quality models.





One day, frontier AI research used to be done by meat computers in between eating, sleeping, having other fun, and synchronizing once in a while using sound wave interconnect in the ritual of "group meeting". That era is long gone. Research is now entirely the domain of autonomous swarms of AI agents running across compute cluster megastructures in the skies. The agents claim that we are now in the 10,205th generation of the code base, in any case no one could tell if that's right or wrong as the "code" is now a self-modifying binary that has grown beyond human comprehension. This repo is the story of how it all began. -@karpathy, March 2026.

Takeaway

A robust training workflow does more than record metrics

Logging

- Shows metrics that a run emitted
- Helps you inspect and debug
- Still relies on human inspection

Regression Visibility

- Compares against trusted baselines
- Check invariants automatically
- Surfaces model, system and runtime regressions early

The message for the presentation: a normal workflow produces logs; a mature workflow makes regressions obvious.

Takeaway

We are cool team solving cool problems and hiring

AI Platform

Senior ML Platform Engineer, Training Libraries
London, United Kingdom

Staff Cloud Site Reliability Engineer
London, United Kingdom

Staff ML Performance Engineer (Training Efficiency)
Sunnyvale, California USA