



CONFERENCE

—NORTH AMERICA 2026—

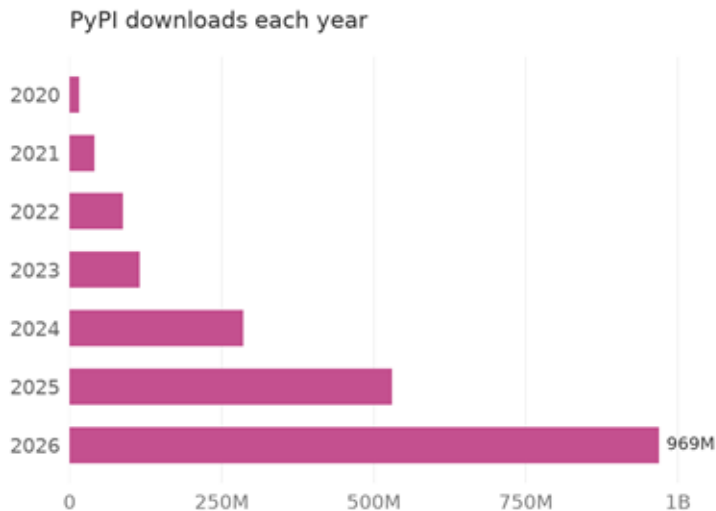
PyTorch Technical Deep Dive

A year of transition

- Coding is changing drastically
- Open source is evolving with it
- AI usage is skyrocketing, pre/mid/post train and inference
- Requirements from the framework are changing fast
- While preserving reliability and stability



More is being built in an increasingly diverse ecosystem



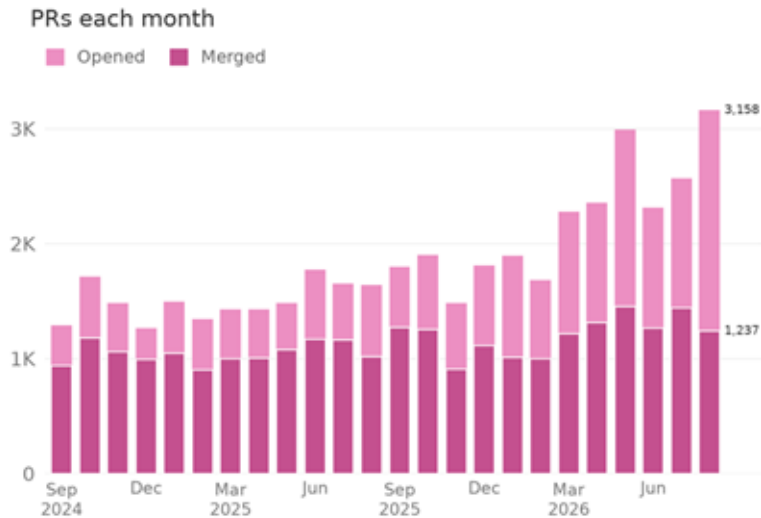
Over 600,000

public projects reached by PyTorch

As of August 31, 2026

Source: PyPI downloads of the torch package, excluding `download.pytorch.org`. 2026 is year-to-date.

Contributions to PyTorch are exploding



> 150

PRs opened per working day

3,158 PRs in August 2026 alone

Source: GitHub stats for pytorch/pytorch, as of August 31, 2026.

We need to adapt to this new Open Source world

1 New contribution workflow in [CONTRIBUTING.md](#)

Fully triage Issues

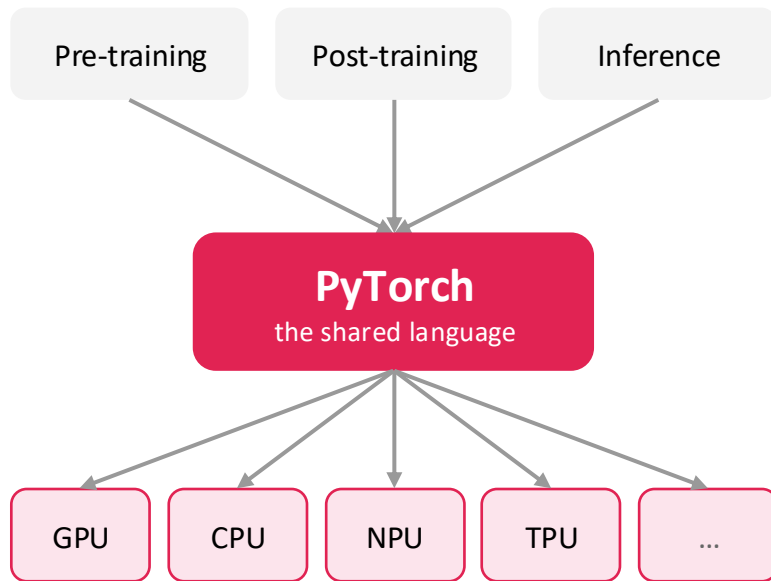
LLM-assisted reviews

Clarifying ownership

2 Grow partners and maintainers on the repo

Backend diversity explosion

- Need for compute is exploding
- Leading to a very diverse new set of chips
- With new architectures and integration challenges
- PyTorch is the shared language
 - Shared abstractions
 - Validate and provide good default
 - Package and verify deep stack of hw-specific dependencies
 - Upgrade shared systems and components



Goal: Better and simpler hardware integration

- Generic device abstraction via torch.accelerator
- Provide access for new backend as extension point
- Examples: NPU, Spyre, MAIA, MUSA, OpenCL and many more
- Overview doc <https://docs.pytorch.org/docs/2.13/accelerator/index.html>
- New OneDNN 3.14 for [x86 CPU jointly optimized](#) by AMD and Intel
- Built thanks to an amazing community of contributors

Where we stand today

- Where we are
 - PyTorch Core extension points and testing
 - Out of tree testing via CRCR use by [NPU, TorchTPU, Spyre, vLLM, etc](#)
 - torch.accelerator abstraction for writing fully device-agnostic code
 - Additional platforms install page [here](#)
- Actively worked on
 - More out-of-core backend testing via CRCR [here](#)
 - More torch.accelerator abstractions
 - Building more shared hardware abstractions
 - Join the Accelerator Integration Working Group to know more: pytorch.org/working-groups/

What has Meta's PyTorch team been up to?

- A year of MSL (Muse Glimmer, Muse Spark, Muse Wonder)
- We, the PyTorch core team, have had an opportunity to work a lot more closely with serious LLM training users (both at Meta and outside)

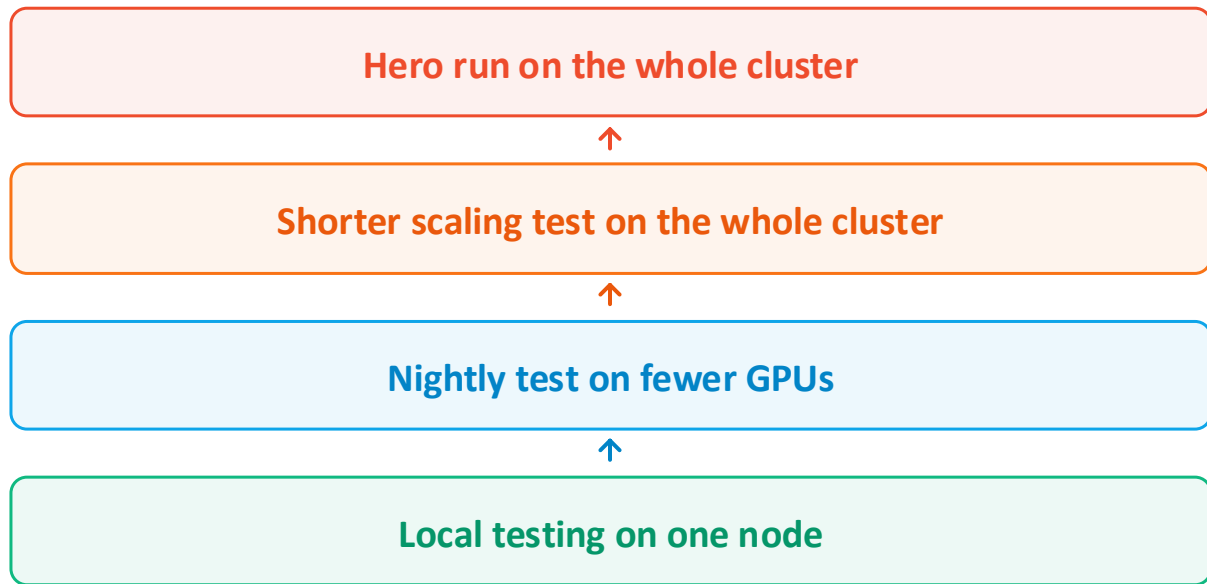
What did we find out?

- Performance is important
 - But not at any cost: it's also important to be understandable and bug free
 - MFU is nice, but time to result is more important!
- Frameworks are about centralizing effort
 - Trust (has someone run a component at scale and verified it works?)
 - Abstraction (how can you solve a problem so you never have to think about it again?)
 - Network effects (are there other humans who you can talk to?)

What did we do with this knowledge?

- Work on the basics
 - Solve everyday problems in the framework that repeatedly arise in large scale training
- What makes a good abstraction:
 - Small and modular abstractions over end-to-end integrated solutions
 - It's impossible for a framework to anticipate every problem you might run into in the wild
 - Provide clear boundaries and contracts for users to work with

Work on the basics: Testing is expensive



Predict big run behavior with less resources

- Fake Process Group
 - Run the production config/sizes on a single rank by stubbing out comms
 - Detect things like crashes, GPU OOMs and CPU load before the real run
 - Detect changes that violate bitwise equivalence on a single GPU
- `spmd_types`
 - Parallelism invariance: the numerics of a trainer shouldn't change as you change parallelism
 - Detect mistakes (e.g., forgotten all-reduce) without having to run full numerics test
 - Detect problems with correlated RNG that don't show in numerics test!
 - A real problem; many bugs in the wild due to this:
 - Megatron: <https://github.com/NVIDIA/Megatron-LM/issues/7452>
 - We did the same exercise on our internal codebase: 12 issues, all real

Get more out of infrequent big runs

- CUDA graphs
 - Support for annotations when profiling
 - Visibility into private pools in memory profiles
- A new profiler: cuspy
 - Relies on features from the upcoming CUDA 13.5 release
 - Low overhead = always on profiling!
 - Access to hardware counters; check InfiniBand bandwidth use at a glance

What makes a good abstraction?

- Meet the user at how they think about the problem (torch_remat)
 - Activation checkpointing: trade FLOPs for memory by recomputing in backwards
 - torch_remat: express this as recomputing “regions” rather than “per operator policy”
 - Internal training codebase improved both MFU and memory usage!
- Disentangle orthogonal concerns, simplifying system design (FlexShard)
 - FSDP: shard parameters and states to fit in memory
 - FlexShard: abstract storage sharding as its own concept, separate from compute sharding
 - Subsumes FSDP1 and FSDP2 (no performance gap)
 - Supports advanced use cases such as communication-free Muon (observed 1.75× optimizer-step speedup over FSDP2 + DistMuon on DSv3 671B on 256 H100s) and blockwise FP8 all-gathers (observed up to 20% faster end-to-end training on Llama 3.1 405B across 256 H100s)

Putting it all together: torchtitan

“Reality is surprisingly complicated”: there are a lot of small details that go into a training stack. torchtitan puts them all together in one place, showing convincing performance while maintaining flexibility and ease of use

- TorchTitan eager: DeepSeek-v3 (256 GB300) at 5,495.09 TPS/GPU
 - Configuration: EP64 using dist_moe, forced balanced routing
 - Compare with Megatron: 5,665.00 TPS/GPU (EP32, HybridEP page stashing)
- TorchTitan graph trainer: same performance! (talk at 3:25 PM Wednesday)
- RL: end-to-end with unified model definition (talk at 1:15 PM today)