

Google open-sources TorchTPU, a native PyTorch backend for its TPUs, with Meta

Source copy kept by wheresthe.ai for the story "Google open-sources TorchTPU, a native PyTorch backend for its TPUs, with Meta" (<https://wheresthe.ai/d/intl-google-open-sources-torchtpu-software-for-pytorch-on-it-2026-10/>).

Source: <https://cloud.google.com/blog/topics/ai-infrastructure/torchtpu-is-now-open-source-run-native-pytorch-on-tpus/>

Text saved 21 Oct 2026 (the page as published).

SHA-256 of the saved text: efdcacad3a67e3389b19a88fd4c1dff61e7832968fca7a0f50735e800f1a7f9a

AI infrastructure

TorchTPU is now open source: Run native PyTorch on TPUs

October 20, 2026

Mark Lohmeyer VP & GM, AI & Computing Infrastructure

Bill Jia Vice President, Engineering

Today at PyTorch Conference 2026, we announced a range of new products and integrations designed to help you run PyTorch effectively on Google Cloud or in your own infrastructure. Notably, TorchTPU, our native PyTorch stack for Google TPUs, is now available as open source on GitHub. With TorchTPU, TPUs join GPUs and CPUs as a native option for PyTorch. You can now train, serve, and deploy models on TPUs using the tools you already rely on, including vLLM, SGLang, Llm-d, TorchTitan, Ray, TorchRec, Hugging Face, LiteRT. Ultimately, this provides our customers choice across a broad range of workload-optimized computing options including TPUs, GPUs, and CPUs, with a consistent, open software and operational model across the full portfolio. Run PyTorch code natively on TPUs Co-developed with the PyTorch team at Meta, TorchTPU lets you run existing PyTorch workloads on TPUs with minimal code changes, using the familiar APIs. TorchTPU lets you debug your models with eager mode, then transition to production-level performance via torch.compile to optimize TPU utilization and reduce memory overhead. By supporting standard PyTorch APIs, advanced diagnostics, and native distributed primitives out of the box, TorchTPU unlocks high performance and operational efficiency from day one.

" At Meta, advancing the frontier of AI requires immense compute scale and deep hardware diversity . We cannot be constrained by a single hardware architecture. TorchTPU is a massive step forward — allowing us to co-develop hardware flexibly with the core machine learning systems team while supporting major LLM, Ads, and Recommendation systems workloads on key production use cases. It enables us to deploy our PyTorch workloads to Google TPUs, provide direct feedback to the engineering teams, and build the ecosystem flexibility everyone needs. "- Jana van Greunen, Director of PyTorch Engineering, Meta

We also partnered with Hugging Face so that popular open-weight models across the Hugging Face Hub run natively on TorchTPU. In Transformers v5, deploying models to TPUs is now as simple as passing device_map="tpu" . The Diffusers library runs natively on TPUs with support for eager mode, torch.compile, and tensor parallelism to scale large generative models across multiple accelerators. To maintain consistent reliability, we also built dedicated CI/CD pipelines for continuous nightly testing of the Hugging Face Transformers suite on TPU infrastructure.

"Hugging Face is committed to being an open, neutral platform where AI builders can choose the models, frameworks and hardware that work best for them. Hardware diversity is essential to making open-source AI accessible to everyone. Our collaboration with Google to bring native TorchTPU support to our core libraries, including Transformers, puts that commitment into practice. Together, we're making it easier for the community to run popular models from the Hub efficiently on Google Cloud TPUs using familiar PyTorch workflows. Giving developers more choice in how and where they build helps the entire open-source ecosystem grow." - Jeff Boudier, VP Product, Hugging Face

Optimizing inference As reasoning and agentic models scale, inference cost and latency become the primary bottlenecks in production. We've integrated TorchTPU into widely used open-source serving engines and paired them with Kubernetes-native fleet routing across TPUs and GPUs to help you optimize throughput per dollar. This foundation allows developers to scale workloads across diverse hardware while minimizing the friction of platform-

specific tuning. To bring these framework-level enhancements to life in production, we have collaborated with leading open-source communities and industry partners. In the sections below, we explore how these partnerships are driving breakthrough efficiency across high-throughput serving with vLLM, structured agentic execution with SGLang, and large-scale distributed fleet orchestration with IIm-d.

vLLM: High-throughput serving and memory efficiency

Developed in partnership with Inferact, the native PyTorch vLLM backend on TorchTPU enables developers to deploy open-weight models directly on TPUs with optimized, production-ready serving features and custom Pallas kernels. This integration supports continuous batching, quantization, and multi-chip parallelism on TPU hardware, while preserving upstream worker models, schedulers, and API servers to improve cost-performance ratios, without requiring changes to existing inference stacks.

" Our mission at Inferact has always been to make high-performance inference accessible, reliable, and cost-effective across the entire AI ecosystem. We are thrilled to partner on the launch of TorchTPU and bring first-class support for vLLM to TPUs. By bridging PyTorch's optimized TPU capabilities directly into vLLM, developers can immediately run production workloads on TPUs with the high throughput, low latency, and memory efficiency they expect from our stack. " - Simon Mo, CEO and co-founder, Inferact

SGLang: Accelerating agentic and structured workloads

Developed in partnership with RadixArk, our native PyTorch backend for SGLang accelerates highly structured and agentic workloads on TPUs. SGLang uses RadixAttention to reuse KV cache across requests, allowing repeated prompt prefixes to skip recomputation entirely. SGLang-TorchTPU preserves upstream worker models, schedulers, and API servers, using a disaggregation transfer backend that also acts as the I/O backend for HiCache to optimize latency and throughput for complex multi-step generation.

" At RadixArk, TorchTPU has supercharged our development velocity as we roll out SGLang support across diverse silicon. It's given us a painless path to native TPU acceleration, passing massive efficiency gains and flexible compute choices directly down to our customers. " - Banghua Zhu, CTO and co-founder, RadixArk

IIm-d for distributed LLM inference

To run inference engines across large fleets, we are releasing IIm-d 1.0 with Red Hat, IBM, and a community of more than 500 contributors and 25 organizations. As a Cloud Native Computing Foundation (CNCF) Sandbox project, IIm-d connects two open-source communities: PyTorch, where vLLM and SGLang serve models, and Kubernetes, where distributed systems run and scale in production. The IIm-d router sends each request to the replica that already holds its context, using prefix-cache-aware routing. It also splits compute-heavy prefill from memory-heavy decode. IIm-d 1.0 makes that ready for production with multi-tenant flow control, router high availability, stable v1 APIs, and session-aware routing for agents. On Google Kubernetes Engine (GKE), you can have the choice of running on either GPUs or Google TPUs, with a common IIm-d endpoint, and choose the hardware that fits each workload.

" At Zenteq, we use GKE Inference Gateway powered by IIm-d to accelerate synthetic data generation for our sovereign Scientific Foundation Models. By leveraging load-aware and prefix-cache-aware routing, we have achieved over 4x higher aggregate inference throughput, increasing daily generated token capacity from 1.47 billion to 5.97 billion tokens per 1,000 TPU v6e accelerators. The open-source foundation of IIm-d, integrated with GKE, enables us to efficiently serve multiple model families at scale without developing and maintaining a custom inference scheduler. " - Prof. Sashikumaar Ganesan, Founder, Zenteq.ai

Training at scale

Scaling PyTorch training across thousands of accelerators requires fault-tolerant, compile-aware distributed frameworks. We have integrated TorchTPU into the PyTorch community's core training, reinforcement learning (RL), and orchestration frameworks.

TorchTitan: Distributed high performance training at scale

TorchTitan is the PyTorch-native reference implementation for scalable distributed training and features a production-grade integration with TorchTPU. By connecting to PyTorch's torch.compile and TorchTitan's compile friendly SimpleFSDP, specialized Pallas kernels, and graph-level optimizations, developers can build and train models on TPUs with high efficiency at scale.

TorchTitan provides recipes for pre-training large dense models (such as Llama 3 70B/405B, Qwen 3 8B/32B, and MuseGlimmer 30B), Mixture-of-Experts (MoE) such as Qwen 3 235B, and diffusion transformers (such as FLUX.1 12B), and has been validated running with up to 8000 Ironwood TPUs (TPU v7x). Beyond initial pre-training, TorchTitan serves as the scalable trainer inside post-training pipelines like VeRL for Group Relative Policy Optimization (GRPO) and RL alignment. To test large-scale efficiency on demanding open architectures, TorchTPU's training paths via TorchTitan demonstrated up to 52% Model Flops Utilization (MFU) on Qwen 3 8B, showing near-linear scaling and stability across up to 8,192 Ironwood TPUs.

VeRL: Scalable post-training and reinforcement learning in preview

RL is a foundational technique for frontier labs that are advancing model capabilities, as well as for enterprises customizing models for domain-specific tasks. Yet, orchestrating distributed RL workloads remains one of the most complex challenges in modern AI infrastructure. Bridging that gap, VeRL has quickly become one of the most popular RL frameworks, supporting everything from verifiable math and coding rewards, to multi-turn agentic tool use at massive scale.

To help expand hardware choice for this thriving community, we upstreamed a TPU backend and an end-to-end reference recipe to VeRL. Built on a disaggregated trainer-sampler architecture designed to maximize training efficiency, our GRPO reference recipe on mathematical reasoning datasets demonstrates how developers can deploy VeRL workloads on TPUs while preserving their existing, hardware-agnostic model and algorithm code.

Ray: Orchestrating heterogeneous compute

We've also added native TorchTPU support to Ray , enabling you to orchestrate PyTorch data processing, training, RL, and serving across CPUs, GPUs, and TPUs within a single workflow. With a few lines of resource configuration, you can schedule each stage of a heterogeneous AI pipeline onto the accelerator best suited for the job.

" At Ray Summit we talked about the PARK stack, PyTorch, AI models, Ray, and Kubernetes, because that's what teams are building production AI on today. TorchTPU embeds Google TPUs directly in that industry-standard stack. We've also worked closely with the Google team to support TPUs as a first-class accelerator in Ray. Coupled with native PyTorch support, teams can now seamlessly develop, scale, and monitor their workloads on TPUs. " - Edward Oakes, Chief Architect, Anyscale

Deployment flexibility and operational efficiency

True portability requires standardizing deployment tools across different hardware profiles, including edge devices and cloud accelerators.

LiteRT joins the PyTorch Ecosystem

Many PyTorch models need to run on phones, laptops, browsers, and embedded devices. Today, LiteRT (formerly TensorFlow Lite), Google's high-performance on-device AI inference, joins the PyTorch ecosystem.

With LiteRT-Torch , you convert a PyTorch model with a single `litert-torch.convert()` call. It builds on `torch.export()` and Core ATen operators, so your model code stays in PyTorch. For LLMs, LiteRT Torch exports Hugging Face Transformer models to run with LiteRT-LM . The converted models run on Android, iOS, macOS, Windows, Linux, web, and IoT devices, with hardware acceleration across CPUs (via XNNPack), GPUs (via open source ML Drift), and NPUs from major chipsets provided through a single API.

TPU agent suite in early preview

To expand PyTorch adoption on TPUs, we are announcing that the TPU agent suite is in preview on GitHub. Integrating through plug-ins with several coding harness-model combinations, the TPU agent suite helps automate onboarding, diagnostics, user questions, and inference performance tuning for many PyTorch models.

Get started with PyTorch on Google Cloud Together, these releases reinforce our commitment to making the Google TPU software ecosystem an open, developer-friendly environment for the PyTorch community. By bringing native TPU support upstream into the PyTorch stack across TorchTPU, Hugging Face, vLLM, SGLang, Llm-d, TorchTitan, VeRL, and Ray — and contributing LiteRT to the PyTorch Ecosystem — we are giving you the flexibility to build, adapt, and scale AI on your terms. Visit the Google Cloud documentation to get started running PyTorch on TPUs today.

Posted in [AI infrastructure](#) [TPUs](#) [Application Development](#) [Open Source](#)

Related articles

[AI infrastructure Simulate, train, and operate robots with Google Cloud physical AI](#) By Alan Christophe • 6-minute read

[AI infrastructure Announcing IIm-d 1.0: Open distributed inference for agents on Google Cloud TPUs and GPUs](#) By Mark Lohmeyer • 8-minute read

[AI infrastructure Day 0 support for Reflection's Beam on Google Cloud: 5x faster perf with IIm-d and NVIDIA Dynamo](#) By Injae Kwak • 8-minute read

[Healthcare & Life Sciences Architecting autonomy: A design pattern for high-compute, governed agents in biotech](#) By Schneider Larbi • 9-minute read