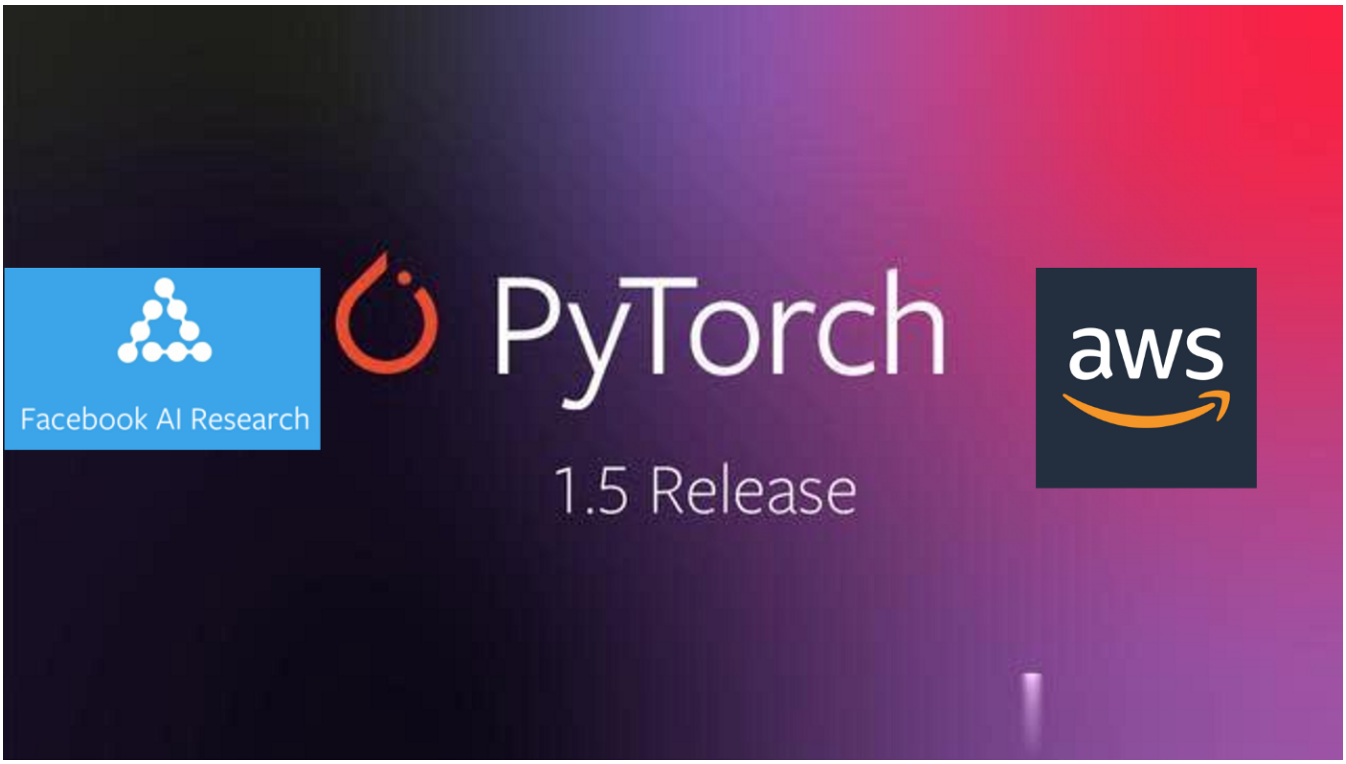
# Facebook and Amazon Bring Two Projects to PyTorch 1.5 that Streamline the Lifecycle of Production-Ready Deep Learning Models

TorchServe and TorchElastic Kubernetes simplify and deployment, training and management of deep learning programs.

Jesus Rodriguez

Apr 22 · 5 min read ★




PyTorch is one of the fastest growing open source projects in the deep learning space. Initially incubated by Facebook, PyTorch rapidly developed a reputation from being an incredibly flexible framework for rapid experimentation and prototyping gaining thousands of fans within the deep learning community. However, PyTorch support for production workload was still lacking its experimentation capabilities. Not surprisingly, many organizations started relying on PyTorch to run experiments with machine learning models and then moving to other frameworks like TensorFlow or

Caffe2 for production workloads. As the PyTorch stack evolved, more organizations started going full circle and relying on the new framework for the implementation of production-ready machine learning programs. A few months ago, AI powerhouse OpenAI announced that it was standardizing on PyTorch as the default framework to power its deep learning research work. Outside Facebook, this is arguably the biggest endorsement for PyTorch within the deep learning world. With that rapid growth comes the need of having better capabilities to support production workflows. The newest version of PyTorch includes two new frameworks that simplify the development and deployment of production-ready deep learning models.

The definition of production-ready deep learning workflows encompasses too many aspects. From large-scale, regular training to deployment and management, operating deep learning models in production environments have very distinctive requirements that contrast with experimental lab settings. In the case of PyTorch, the recent release addresses two key aspects of production-ready deep learning programs: model serving and large-scale training.

## TorchServe

Model serving is the process of deploying new algorithms and experiments to production environments and enable the corresponding API to consume them. Model serving has become such an important part of the lifecycle of deep learning models that has created a new category of products: model servers. Projects such as TensorFlow Serving and the Multi Model Server as notorious examples of model servers. PyTorch has been rapidly building up its model serving capabilities which became front and center of its latest release.

TorchServe is a collaboration between AWS and Facebook intended to streamline the production deployment of PyTorch models without the need of writing code. Functionally, TorchServe allow data scientists to bring their models to production environments such as Amazon SageMaker, container services, and Amazon Elastic Compute Cloud (EC2) without having to modify the models. TorchServe includes a series of capabilities that simplify the deployment and management of deep learning models in production environments. Some of the most notable features of TorchServe include:

- *Default handlers for common use cases (e.g., image segmentation, text classification) as well as the ability to write custom handlers for other use cases*

- *Model versioning, the ability to run multiple versions of a model at the same time, and the ability to roll back to an earlier version*
- *Robust management capability, allowing full configuration of models, versions, and individual worker threads via command line, config file, or run-time API*
- *Automatic batching of individual inferences across HTTP requests*
- *Logging including common metrics, and the ability to incorporate custom metrics*
- *Ready-made Dockerfile for easy deployment*
- *HTTPS support for secure deployment*

The current version of TorchServe includes a built-in web server that can be run from a command line. This command line call takes in the single or multiple models you want to serve, along with additional optional parameters controlling the port, host, and logging. TorchServe supports running custom services to handle the specific inference handling logic.

Using TorchServe is super simple. Let's assume that we would like to deploy a model archive "targetmodel.mar" from a model store "model_store". The following command line will serve that model to the TorchServe model server.

```
torchserve --start --model-store model_store --models
targetmodel.mar
```

Similarly, we can load all the models available in model_store while starting torchserve:

```
torchserve --model-store /models --start --models all
```

The integration with different cloud environments, especially in the AWS cloud, is another tangible benefit of TorchServe. This feature allows machine learning engineers to leverage the sophisticated toolset of platforms such as AWS SageMaker without sacrificing the PyTorch development experience.

## TorchElastic Kubernetes

Large scale training is another key characteristic of production-ready deep learning programs. In the PyTorch ecosystem, TorchElastic has been a popular library for training large scale deep neural networks. In the latest release, AWS and Facebook engineers collaborated in expanding TorchElastic's capabilities by integrating it with Kubernetes in the form of the TorchElastic Controller for Kubernetes(TECK).

Conceptually, TorchElastic enables the training large-scale deep learning models where it's critical to scale compute resources dynamically based on availability. It provides the primitives and interfaces for you to write your PyTorch job in such a way that it can be run on multiple machines with elasticity. That is, your distributed job is able to start as soon as min number of workers are present and allowed to grow up to max number of workers without being stopped or restarted.

Part of the challenge of large-scale training jobs is to distribute the workloads across a large number of GPU/CPU instances in a concurrent and fault-tolerant manner. Kubernetes excels at workload distribution tasks. TheTECK expands TorchElastic with a Kubernetes interface to distribute training jobs. Specifically, TECK is a native Kubernetes implementation of the PyTorch Elastic interface that automatically manages the lifecycle of the Kubernetes pods and services required for TorchElastic training. Using a simple custom resource definition, you specify your Elastic compatible training image along with the desired, min, and max number of workers. It allows you to start mission critical distributed training jobs on Kubernetes clusters with a portion of the requested compute resources, and dynamically scale later as more resources become available, without having to stop and restart the jobs.

Leveraging Kubernetes for training PyTorch models would help to reduce distributed training time and costs by limiting idle cluster resources and enabling job recovery from failures. Many data science teams were already using Kubernetes and PyTorch together but the training angle enabled by TECK is certainly unique.

Both TorchServe and TECK are welcomed additions to the PyTorch stack that should improve its usage for production-ready workloads. Furthermore, the AWS-Facebook collaboration could be a first step towards making AWS the preferred cloud runtime for running PyTorch programs. Hopefully, this is an initial step in a fruitful collaboration between the AWS and PyTorch teams.

Machine Learning Deep Learning Data Science Artificial Intelligence Invector Labs