

---

# PyTorch: An Imperative Style, High-Performance Deep Learning Library

---

**Adam Paszke**  
University of Warsaw  
adam.paszke@gmail.com

**Sam Gross**  
Facebook AI Research  
sgross@fb.com

**Francisco Massa**  
Facebook AI Research  
fmassa@fb.com

**Adam Lerer**  
Facebook AI Research  
alerer@fb.com

**James Bradbury**  
Google  
jekbradbury@gmail.com

**Gregory Chanan**  
Facebook AI Research  
gchanan@fb.com

**Trevor Killeen**  
Self Employed  
killeent@cs.washington.edu

**Zeming Lin**  
Facebook AI Research  
zlin@fb.com

**Natalia Gimelshein**  
NVIDIA  
ngimelshein@nvidia.com

**Luca Antiga**  
Orobix  
luca.antiga@orobix.com

**Alban Desmaison**  
Oxford University  
alban@robots.ox.ac.uk

**Andreas Köpf**  
Xamla  
andreas.koepf@xamla.com

**Edward Yang**  
Facebook AI Research  
ezyang@fb.com

**Zach DeVito**  
Facebook AI Research  
zdevito@cs.stanford.edu

**Martin Raison**  
Nabla  
martinraison@gmail.com

**Alykhan Tejani**  
Twitter  
atejani@twitter.com

**Sasank Chilamkurthy**  
Qure.ai  
sasankchilamkurthy@gmail.com

**Benoit Steiner**  
Facebook AI Research  
benoitsteiner@fb.com

**Lu Fang**  
Facebook  
lufang@fb.com

**Junjie Bai**  
Facebook  
jbai@fb.com

**Soumith Chintala**  
Facebook AI Research  
soumith@gmail.com

## Abstract

Deep learning frameworks have often focused on either usability or speed, but not both. PyTorch is a machine learning library that shows that these two goals are in fact compatible: it provides an imperative and Pythonic programming style that supports code as a model, makes debugging easy and is consistent with other popular scientific computing libraries, while remaining efficient and supporting hardware accelerators such as GPUs.

In this paper, we detail the principles that drove the implementation of PyTorch and how they are reflected in its architecture. We emphasize that every aspect of PyTorch is a regular Python program under the full control of its user. We also explain how the careful and pragmatic implementation of the key components of its runtime enables them to work together to achieve compelling performance.

We demonstrate the efficiency of individual subsystems, as well as the overall speed of PyTorch on several common benchmarks.

# 1 Introduction

With the increased interest in deep learning in recent years, there has been an explosion of machine learning tools. Many popular frameworks such as Caffe [1], CNTK [2], TensorFlow [3], and Theano [4], construct a static dataflow graph that represents the computation and which can then be applied repeatedly to batches of data. This approach provides visibility into the whole computation ahead of time, and can theoretically be leveraged to improve performance and scalability. However, it comes at the cost of ease of use, ease of debugging, and flexibility of the types of computation that can be represented.

Prior work has recognized the value of dynamic eager execution for deep learning, and some recent frameworks implement this define-by-run approach, but do so either at the cost of performance (Chainer [5]) or using a less expressive, faster language (Torch [6], DyNet [7]), which limits their applicability.

However, with careful implementation and design choices, dynamic eager execution can be achieved largely without sacrificing performance. This paper introduces PyTorch, a Python library that performs immediate execution of dynamic tensor computations with automatic differentiation and GPU acceleration, and does so while maintaining performance comparable to the fastest current libraries for deep learning. This combination has turned out to be very popular in the research community with, for instance, 296 ICLR 2019 submissions mentioning PyTorch.

# 2 Background

Four major trends in scientific computing have become increasingly important for deep learning.

First, starting in the 1960s, the development of domain specific languages such as APL [8], MATLAB [9], R [10] and Julia [11], turned multidimensional arrays (often referred to as tensors) into first-class objects supported by a comprehensive set of mathematical primitives (or operators) to manipulate them. Separately, libraries such as NumPy [12], Torch [6], Eigen [13] and Lush [14] made **array-based programming** productive in general purpose languages such as Python, Lisp, C++ and Lua.

Second, the development of **automatic differentiation** [15] made it possible to fully automate the daunting labor of computing derivatives. This made it significantly easier to experiment with different machine learning approaches while still allowing for efficient gradient based optimization. The autograd [16] package popularized the use of this technique for NumPy arrays, and similar approaches are used in frameworks such as Chainer [5], DyNet [7], Lush [14], Torch [6], Jax [17] and Flux.jl [18].

Third, with the advent of the free software movement, the scientific community moved away from closed proprietary software such as Matlab [9], and towards the **open-source Python ecosystem** with packages like NumPy [12], SciPy [19], and Pandas [20]. This fulfilled most of the numerical analysis needs of researchers while allowing them to take advantage of a vast repository of libraries to handle dataset preprocessing, statistical analysis, plotting, and more. Moreover, the openness, interoperability, and flexibility of free software fostered the development of vibrant communities that could quickly address new or changing needs by extending the existing functionality of a library or if needed by developing and releasing brand new ones. While there is a rich offering of open-source software for neural networks in languages other than Python, starting with Lush [14] in Lisp, Torch [6] in C++, Objective-C and Lua, EBLearn [21] in C++, Caffe [1] in C++, the network effects of a large ecosystem such as Python made it an essential skill to jumpstart one's research. Hence, since 2014, most deep learning frameworks converged on a Python interface as an essential feature.

Finally, the availability and commoditization of general-purpose massively parallel hardware such as GPUs provided the computing power required by deep learning methods. Specialized libraries such as cuDNN [22], along with a body of academic work (such as [23] and [24]), produced a set of high-performance reusable deep learning kernels that enabled frameworks such as Caffe [1], Torch7 [25], or TensorFlow [3] to take advantage of these **hardware accelerators**.

PyTorch builds on these trends by providing an array-based programming model accelerated by GPUs and differentiable via automatic differentiation integrated in the Python ecosystem.