The Journal of Open Source Software

xTorch: A High-Level C++ Extension Library for PyTorch (LibTorch)

kamran saberifard [1*]

1 Aryorithm * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Vissarion Fisikopoulos

Reviewers:

- @justusschock
- @arishs24

Submitted: 08 April 2025
Published: unpublished

Summary

PyTorch's C++ library (LibTorch) emerged as a powerful way to use PyTorch outside Python, but after 2019 it became challenging for developers to use it for end-to-end model development. Early on, LibTorch aimed to mirror the high-level Python API, yet many convenient abstractions and examples never fully materialized or were later removed. As of 2020, the C++ API had achieved near feature-parity with Python's core operations, but it lagged in usability and community support. Fewer contributors focused on C++ meant that only low-level building blocks were provided, with high-level components (e.g. ready-made network architectures, datasets) largely absent. This left C++ practitioners to rewrite common tools from scratch – implementing standard models or data loaders manually – which is time-consuming and error-prone. Another factor was PyTorch's emphasis on the Python-to-C++ workflow. The official recommended path for production was to prototype in Python, then convert models to TorchScript for C++ deployment. This approach deprioritized making the pure C++ experience as friendly as Python's. As a result, developers who preferred or needed to work in C++ (for integration with existing systems, performance, or deployment constraints) found LibTorch cumbersome. Simple tasks like data augmentation (e.g. random crops or flips) had no built-in support in LibTorch C++. Defining neural network modules in C++ involved boilerplate macros and manual registration, an awkward process compared to Python's concise syntax. Crucial functionality for model serialization was limited – for instance, LibTorch could load Python-exported models but not easily export its own models to a portable format. xTorch was created to address this gap. It is a C++ library that extends LibTorch with the high-level abstractions and utilities that were missing or removed after 2019. By building on LibTorch's robust computational core, xTorch restores ease-of-use without sacrificing performance. The motivation is to empower C++ developers with a productive experience similar to PyTorch in Python – enabling them to build, train, and deploy models with minimal fuss. In essence, xTorch revives and modernizes the "batteries-included" ethos for C++ deep learning, providing an all-in-one toolkit where the base library left off.

Motivation

PyTorch's C++ library (LibTorch) emerged as a powerful way to use PyTorch outside Python, but after 2019 it became challenging for developers to use it for end-to-end model development. Early on, LibTorch aimed to mirror the high-level Python API, yet many convenient abstractions and examples never fully materialized or were later removed.

As of 2020, the C++ API had achieved near feature-parity with Python's core operations, but it lagged in usability and community support. Fewer contributors focused on C++ meant that only low-level building blocks were provided, with high-level components (e.g. ready-made network architectures, datasets) largely absent. This left C++ practitioners to rewrite common tools from scratch – implementing standard models or data loaders manually – which is time-consuming and error-prone.

Another factor was PyTorch's emphasis on the Python-to-C++ workflow. The official recommended path for production was to prototype in Python, then convert models to TorchScript for C++ deployment. This approach deprioritized making the pure C++ experience as friendly as Python's.

As a result, developers who preferred or needed to work in C++ (for integration with existing systems, performance, or deployment constraints) found LibTorch cumbersome. Simple tasks like data augmentation (e.g. random crops or flips) had no built-in support in LibTorch C++. Defining neural network modules in C++ involved boilerplate macros and manual registration, an awkward process compared to Python's concise syntax. Crucial functionality for model serialization was limited – for instance, LibTorch could load Python-exported models but not easily export its own models to a portable format.

xTorch was created to address this gap. It is a C++ library that extends LibTorch with the high-level abstractions and utilities that were missing or removed after 2019. By building on LibTorch's robust computational core, xTorch restores ease-of-use without sacrificing performance. The motivation is to empower C++ developers with a productive experience similar to PyTorch in Python – enabling them to build, train, and deploy models with minimal fuss. In essence, xTorch revives and modernizes the "batteries-included" ethos for C++ deep learning, providing an all-in-one toolkit where the base library left off.

Design and Architecture

xTorch is architected as a thin layer on top of LibTorch's C++ API, carefully integrating with it rather than reinventing it. The design follows a modular approach, adding a higher-level API that wraps around LibTorch's lower-level classes. At its core, xTorch relies on LibTorch for tensor operations, autograd, and neural network primitives – effectively using LibTorch as the computational engine. The extended library then introduces its own set of C++ classes that encapsulate common patterns (model definitions, training loops, data handling, etc.), providing a cleaner interface to the developer.

Architecture Layers

LibTorch Core (Bottom Layer): Provides torch::Tensor, torch::autograd, torch::nn, optimizers, etc. Extended Abstraction Layer (Middle): Simplified classes inheriting from LibTorch core (e.g., ExtendedModel, Trainer). User Interface (Top Layer): Intuitive APIs and boilerplate-free interaction.

Modules

Model Module: High-level model class extensions. Data Module: Enhanced datasets and DataLoader. Training Module: Training logic, checkpointing, metrics. Utilities Module: Logging, device helpers, summaries.

Features and Enhancements

High-Level Model Classes: XTModule, prebuilt models like ResNetExtended, XTCNN. Simplified Training Loop (Trainer): Full training abstraction with callbacks and metrics. Enhanced Data Handling: ImageFolderDataset, CSVDataset, OpenCV-backed support. Utility Functions: Logging, metrics, summary, device utils. Extended Optimizers: AdamW, RAdam, schedulers, learning rate strategies. Model Serialization & Deployment: save_model(), export_to_jit(), inference helpers.

Statement of need

xTorch addresses the lack of high-level APIs in LibTorch for C++ developers, which is critical for high-performance machine learning, robotics, embedded applications, and large-scale deployment scenarios. By reintroducing high-level utilities that were deprecated in the Python API post-2019, xTorch enables C++ developers to build, train, evaluate, and deploy models more intuitively and efficiently.

C++ remains a critical language for high-performance machine learning systems, robotics, embedded applications, and large-scale deployment. However, PyTorch's C++ frontend (LibTorch) is difficult to use on its own due to the lack of high-level APIs, forcing users to write verbose and repetitive code.

xTorch was created to fill this gap by wrapping LibTorch with practical utilities such as `Trainer`, `XTModule`, `DataLoader`, and `export_to_jit()`. These abstractions drastically reduce boilerplate, increase accessibility, and allow developers to build, train, and deploy models entirely in C++. Unlike other frameworks that require switching to Python or writing extensive C++ glue code, xTorch makes the entire ML workflow intuitive and modular in C++.

Functionality

xTorch provides:

- High-level neural network module definitions (e.g., XTModule, ResNetExtended, XTCNN)
- A simplified training loop with the Trainer class, handling loss computation, metrics, and callbacks
- Enhanced data handling with ImageFolderDataset, CSVDataset, and OpenCV-backed transformations
- Utility functions for logging, metrics computation, and device management
- Extended optimizers like AdamW, RAdam, and learning rate schedulers
- Model serialization and TorchScript export helpers (save_model(), export_to_jit())
- Inference utilities for loading models and making predictions

The library is modular and extensible, built on top of LibTorch, and supports both CPU and CUDA devices.

Example Use

```cpp
// Example: CNN Training Pipeline
auto trainData = xt::datasets::ImageFolder("data/train", xt::transforms::Compose({
    xt::transforms::Resize({224, 224}),
    xt::transforms::ToTensor(),
    xt::transforms::Normalize({0.5, 0.5, 0.5}, {0.5, 0.5, 0.5})
}));

auto trainLoader = xt::data::DataLoader(trainData, 64, true);

auto model = xt::models::ResNet18(10);
auto optimizer = xt::optim::Adam(model.parameters(), 1e-3);
auto criterion = xt::loss::CrossEntropyLoss();

xt::Trainer trainer;
trainer.setMaxEpochs(20)
       .setOptimizer(optimizer)
       .setCriterion(criterion)
       .fit(model, trainLoader);
```

```
// Export model to TorchScript
xt::export_to_jit(model, "model.pt");
```

Acknowledgements

The xTorch project builds upon the PyTorch (LibTorch) C++ API. Thanks to the open-source contributors to PyTorch for enabling access to their high-performance machine learning framework via C++.

References

- PyTorch C++ API Documentation: https://pytorch.org/cppdocs/
- TorchScript for Deployment: https://pytorch.org/tutorials/advanced/cpp_export.html