C++

> **Note**
>
> If you are looking for the PyTorch C++ API docs, directly go here.

PyTorch provides several features for working with C++, and itâ€™s best to choose from them based on your needs. At a high level, the following support is available:

TorchScript C++ API

TorchScript allows PyTorch models defined in Python to be serialized and then loaded and run in C++ capturing the model code via compilation or tracing its execution. You can learn more in the Loading a TorchScript Model in C++ tutorial. This means you can define your models in Python as much as possible, but subsequently export them via TorchScript for doing no-Python execution in production or embedded environments. The TorchScript C++ API is used to interact with these models and the TorchScript execution engine, including:

- Loading serialized TorchScript models saved from Python
- Doing simple model modifications if needed (e.g. pulling out submodules)
- Constructing the input and doing preprocessing using C++ Tensor API

Extending PyTorch and TorchScript with C++ Extensions

TorchScript can be augmented with user-supplied code through custom operators and custom classes. Once registered with TorchScript, these operators and classes can be invoked in TorchScript code run from Python or from C++ as part of a serialized TorchScript model. The Extending TorchScript with Custom C++ Operators tutorial walks through interfacing TorchScript with OpenCV. In addition to wrapping a function call with a custom operator, C++ classes and structs can be bound into TorchScript through a pybind11-like interface which is explained in the Extending TorchScript with Custom C++ Classes tutorial.

Tensor and Autograd in C++

Most of the tensor and autograd operations in PyTorch Python API are also available in the C++ API. These include:

- `torch::Tensor` methods such as `add` / `reshape` / `clone`. For the full list of methods available, please see: https://pytorch.org/cppdocs/api/classat_1_1_tensor.html
- C++ tensor indexing API that looks and behaves the same as the Python API. For details on its usage, please see: https://pytorch.org/cppdocs/notes/tensor_indexing.html
- The tensor autograd APIs and the `torch::autograd` package that are crucial for building dynamic neural networks in C++ frontend. For more details, please see: https://pytorch.org/tutorials/advanced/cpp_autograd.html

Authoring Models in C++

The "author in TorchScript, infer in C++" workflow requires model authoring to be done in TorchScript. However, there might be cases where the model has to be authored in C++ (e.g. in workflows where a Python component is undesirable). To serve such use cases, we provide the full capability of authoring and training a neural net model purely in C++, with familiar components such as `torch::nn` / `torch::nn::functional` / `torch::optim` that closely resemble the Python API.

- For an overview of the PyTorch C++ model authoring and training API, please see: https://pytorch.org/cppdocs/frontend.html
- For a detailed tutorial on how to use the API, please see: https://pytorch.org/tutorials/advanced/cpp_frontend.html
- Docs for components such as `torch::nn` / `torch::nn::functional` / `torch::optim` can be found at: https://pytorch.org/cppdocs/api/library_root.html

Packaging for C++

For guidance on how to install and link with libtorch (the library that contains all of the above C++ APIs), please see: https://pytorch.org/cppdocs/installing.html. Note that on Linux there are two types of libtorch binaries provided: one compiled with GCC pre-cxx11 ABI and the other with GCC cxx11 ABI, and you should make the selection based on the GCC ABI your system is using.