PyTorch Contribution Guide

PyTorch is a GPU-accelerated Python tensor computation package for building deep neural networks built on tape-based autograd systems.

The PyTorch Contribution Process

The PyTorch organization is governed by :doc:`PyTorch Governance <governance>`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\pytorch-master\docs\source\community\[pytorch-master][docs][source][community]contribution_guide.rst`, **line 10**); *backlink*
>
> Unknown interpreted text role "doc".

The PyTorch development process involves a healthy amount of open discussions between the core development team and the community.

PyTorch operates similar to most open source projects on GitHub. However, if you've never contributed to an open source project before, here is the basic process.

- **Figure out what you're going to work on.** The majority of open source contributions come from people scratching their own itches. However, if you don't know what you want to work on, or are just looking to get more acquainted with the project, here are some tips for how to find appropriate tasks:
  - Look through the issue tracker and see if there are any issues you know how to fix. Issues that are confirmed by other contributors tend to be better to investigate. We also maintain some labels for issues that are likely to be good for new people, e.g., **bootcamp** and **1hr**, although these labels are less well maintained.
  - Join us on Slack and let us know you're interested in getting to know PyTorch. We're very happy to help out researchers and partners get up to speed with the codebase.
- **Figure out the scope of your change and reach out for design comments on a GitHub issue if it's large.** The majority of pull requests are small; in that case, no need to let us know about what you want to do, just get cracking. But if the change is going to be large, it's usually a good idea to get some design comments about it first.
  - If you don't know how big a change is going to be, we can help you figure it out! Just post about it on issues or Slack.
  - Some feature additions are very standardized; for example, lots of people add new operators or optimizers to PyTorch. Design discussion in these cases boils down mostly to, "Do we want this operator/optimizer?" Giving evidence for its utility, e.g., usage in peer reviewed papers, or existence in other frameworks, helps a bit when making this case.
    - **Adding operators / algorithms from recently-released research** is generally not accepted unless there is overwhelming evidence that this newly published work has ground-breaking results and will eventually become a standard in the field. If you are not sure where your method falls, open an issue first before implementing a PR.
  - Core changes and refactors can be quite difficult to coordinate, as the pace of development on PyTorch master is quite fast. Definitely reach out about fundamental or cross-cutting changes; we can often give guidance about how to stage such changes into more easily reviewable pieces.
- **Code it out!**
  - See the technical guide for advice for working with PyTorch in a technical form.
- **Open a pull request.**
  - If you are not ready for the pull request to be reviewed, tag it with [WIP]. We will ignore it when doing review passes. If you are working on a complex change, it's good to start things off as WIP, because you will need to spend time looking at CI results to see if things worked out or not.
  - Find an appropriate reviewer for your change. We have some folks who regularly go through the PR queue and try to review everything, but if you happen to know who the maintainer for a given subsystem affected by your patch is, feel free to include them directly on the pull request. You can learn more about this structure at PyTorch Subsystem Ownership.
- **Iterate on the pull request until it's accepted!**
  - We'll try our best to minimize the number of review roundtrips and block PRs only when there are major issues. For the most common issues in pull requests, take a look at Common Mistakes.
  - Once a pull request is accepted and CI is passing, there is nothing else you need to do; we will merge the PR for you.

Getting Started

Proposing new features

New feature ideas are best discussed on a specific issue. Please include as much information as you can, any accompanying data, and your proposed solution. The PyTorch team and community frequently review new issues and comments where they think they can help. If you feel confident in your solution, go ahead and implement it.

Reporting Issues

If you've identified an issue, first search through the list of existing issues on the repo. If you are unable to find a similar issue, then create a new one. Supply as much information you can to reproduce the problematic behavior. Also, include any additional insights like the behavior you expect.

Implementing Features or Fixing Bugs

If you want to fix a specific issue, it's best to comment on the individual issue with your intent. However, we do not lock or assign issues except in cases where we have worked with the developer before. It's best to strike up a conversation on the issue and discuss your proposed solution. The PyTorch team can provide guidance that saves you time.

Issues that are labeled first-new-issue, low, or medium priority provide the best entrance point are great places to start.

Adding Tutorials

A great deal of the tutorials on pytorch.org come from the community itself and we welcome additional contributions. To learn more about how to contribute a new tutorial you can learn more here: PyTorch.org Tutorial Contribution Guide on Github

Improving Documentation & Tutorials

We aim to produce high quality documentation and tutorials. On rare occasions that content includes typos or bugs. If you find something you can fix, send us a pull request for consideration.

Take a look at the Documentation section to learn how our system works.

Participating in online discussions

You can find active discussions happening on the PyTorch Discussion forum.

Submitting pull requests to fix open issues

You can view a list of all open issues here. Commenting on an issue is a great way to get the attention of the team. From here you can share your ideas and how you plan to resolve the issue.

For more challenging issues, the team will provide feedback and direction for how to best solve the issue.

If you're not able to fix the issue itself, commenting and sharing whether you can reproduce the issue can be useful for helping the team identify problem areas.

Reviewing open pull requests

We appreciate your help reviewing and commenting on pull requests. Our team strives to keep the number of open pull requests at a manageable size, we respond quickly for more information if we need it, and we merge PRs that we think are useful. However, due to the high level of interest, additional eyes on the pull requests are appreciated.

Improving code readability

Improve code readability helps everyone. It is often better to submit a small number of pull requests that touch few files versus a large pull request that touches many files. Starting a discussion in the PyTorch forum here or on an issue related to your improvement is the best way to get started.

Adding test cases to make the codebase more robust

Additional test coverage is appreciated.

Promoting PyTorch

Your use of PyTorch in your projects, research papers, write ups, blogs, or general discussions around the internet helps to raise awareness for PyTorch and our growing community. Please reach out to pytorch-marketing@fb.com for marketing support.

Triaging issues

If you feel that an issue could benefit from a particular tag or level of complexity, comment on the issue and share your opinion. If you feel an issue isn't categorized properly, comment and let the team know.

About open source development

If this is your first time contributing to an open source project, some aspects of the development process may seem unusual to you.

- **There is no way to "claim" issues.** People often want to "claim" an issue when they decide to work on it, to ensure that there isn't wasted work when someone else ends up working on it. This doesn't really work too well in open source, since someone may decide to work on something, and end up not having time to do it. Feel free to give information in an advisory

fashion, but at the end of the day, we will take running code and rough consensus.
- **There is a high bar for new functionality that is added.** Unlike in a corporate environment, where the person who wrote code implicitly "owns" it and can be expected to take care of it at the beginning of its lifetime, once a pull request is merged into an open source project, it immediately becomes the collective responsibility of all maintainers on the project. When we merge code, we are saying that we, the maintainers, can review subsequent changes and make a bugfix to the code. This naturally leads to a higher standard of contribution.

Common Mistakes To Avoid

- **Did you add tests?** (Or if the change is hard to test, did you describe how you tested your change?)
  - We have a few motivations for why we ask for tests:
    1. to help us tell if we break it later
    2. to help us tell if the patch is correct in the first place (yes, we did review it, but as Knuth says, "beware of the following code, for I have not run it, merely proven it correct")
  - When is it OK not to add a test? Sometimes a change can't be conveniently tested, or the change is so obviously correct (and unlikely to be broken) that it's OK not to test it. On the contrary, if a change seems likely (or is known to be likely) to be accidentally broken, it's important to put in the time to work out a testing strategy.
- **Is your PR too long?**
  - It's easier for us to review and merge small PRs. The difficulty of reviewing a PR scales nonlinearly with its size.
  - When is it OK to submit a large PR? It helps a lot if there was a corresponding design discussion in an issue, with sign off from the people who are going to review your diff. We can also help give advice about how to split up a large change into individually shippable parts. Similarly, it helps if there is a complete description of the contents of the PR: it's easier to review code if we know what's inside!
- **Comments for subtle things?** In cases where the behavior of your code is nuanced, please include extra comments and documentation to allow us to better understand the intention of your code.
- **Did you add a hack?** Sometimes, the right answer is a hack. But usually, we will have to discuss it.
- **Do you want to touch a very core component?** To prevent major regressions, pull requests that touch core components receive extra scrutiny. Make sure you've discussed your changes with the team before undertaking major changes.
- **Want to add a new feature?** If you want to add new features, comment your intention on the related issue. Our team tries to comment on and provide feedback to the community. It's better to have an open discussion with the team and the rest of the community before building new features. This helps us stay aware of what you're working on and increases the chance that it'll be merged.
- **Did you touch unrelated code to the PR?** To aid in code review, please only include files in your pull request that are directly related to your changes.

Frequently asked questions

- **How can I contribute as a reviewer?** There is lots of value if community developers reproduce issues, try out new functionality, or otherwise help us identify or troubleshoot issues. Commenting on tasks or pull requests with your environment details is helpful and appreciated.
- **CI tests failed, what does it mean?** Maybe you need to merge with master or rebase with latest changes. Pushing your changes should re-trigger CI tests. If the tests persist, you'll want to trace through the error messages and resolve the related issues.
- **What are the most high risk changes?** Anything that touches build configuration is a risky area. Please avoid changing these unless you've had a discussion with the team beforehand.
- **Hey, a commit showed up on my branch, what's up with that?** Sometimes another community member will provide a patch or fix to your pull request or branch. This is often needed for getting CI tests to pass.

On Documentation

Python Docs

PyTorch documentation is generated from python source using Sphinx. Generated HTML is copied to the docs folder in the master branch of pytorch.github.io, and is served via GitHub pages.

- Site: https://pytorch.org/docs
- GitHub: https://github.com/pytorch/pytorch/tree/master/docs
- Served from: https://github.com/pytorch/pytorch.github.io/tree/master/doc

C++ Docs

For C++ code we use Doxygen to generate the content files. The C++ docs are built on a special server and the resulting files are copied to the https://github.com/pytorch/cppdocs repo, and are served from GitHub pages.

- Site: https://pytorch.org/cppdocs
- GitHub: https://github.com/pytorch/pytorch/tree/master/docs/cpp

- Served from: https://github.com/pytorch/cppdocs

Tutorials

PyTorch tutorials are documents used to help understand using PyTorch to accomplish specific tasks or to understand more holistic concepts. Tutorials are built using Sphinx-Gallery from executable python sources files, or from restructured-text (rst) files.

- Site: https://pytorch.org/tutorials
- GitHub: https://github.com/pytorch/tutorials

Tutorials Build Overview

For tutorials, pull requests trigger a rebuild of the entire site using CircleCI to test the effects of the change. This build is sharded into 9 worker builds and takes around 40 minutes total. At the same time, we do a Netlify build using *make html-noplot*, which builds the site without rendering the notebook output into pages for quick review.

After a PR is accepted, the site is rebuilt and deployed from CircleCI.

Contributing a new Tutorial

PyTorch.org Tutorial Contribution Guide