# Contributing

Welcome to the PyTorch Lightning community! We're building the most advanced research platform on the planet to implement the latest, best practices that the amazing PyTorch team rolls out!

## Main Core Value: One less thing to remember

Simplify the API as much as possible from the user perspective.

Any additions or improvements should minimize the things the user needs to remember.

For example: One benefit of the validation\_step is that the user doesn't have to remember to set the model to .eval().

This helps users avoid all sorts of subtle errors.

## Lightning Design Principles

We encourage all sorts of contributions you're interested in adding! When coding for lightning, please follow these principles.

#### No PyTorch Interference

We don't want to add any abstractions on top of pure PyTorch.

This gives researchers all the control they need without having to learn yet another framework.

#### Simple Internal Code

It's useful for users to look at the code and understand very quickly what's happening.

Many users won't be engineers. Thus we need to value clear, simple code over condensed ninja moves.

While that's super cool, this isn't the project for that :)

#### Force User Decisions To Best Practices

There are 1,000 ways to do something. However, eventually one popular solution becomes standard practice, and everyone follows.

We try to find the best way to solve a particular problem, and then force our users to use it for readability and simplicity.

A good example is accumulated gradients.

There are many different ways to implement it, we just pick one and force users to use it.

A bad forced decision would be to make users use a specific library to do something.

When something becomes a best practice, we add it to the framework. This is usually something like bits of code in utils or in the model file that everyone keeps adding over and over again across projects. When this happens, bring that code inside the trainer and add a flag for it.

#### Simple External API

What makes sense to you may not make sense to others. When creating an issue with an API change suggestion, please validate that it makes sense for others.

Treat code changes the way you treat a startup: validate that it's a needed feature, then add if it makes sense for many people.

#### Backward-compatible API

We all hate updating our deep learning packages because we don't want to refactor a bunch of stuff. In Lightning, we make sure every change we make which could break an API is backwards compatible with good deprecation warnings.

\*\*You shouldn't be afraid to upgrade Lightning :)\*\*

#### Gain User Trust

As a researcher you can't have any part of your code going wrong. So, make thorough tests to ensure that every implementation of a new trick or subtle change is correct.

#### Interoperability

Have a favorite feature from other libraries like fast.ai or transformers? Those should just work with lightning as well. Grab your favorite model or learning rate scheduler from your favorite library and run it in Lightning.

---

## Contribution Types

We are currently looking for help implementing new features or adding bug fixes.

A lot of good work has already been done in project mechanics (requirements/base.txt, setup.py, pep8, badges, ci, etc...) so we're in a good state there thanks to all the early contributors (even pre-beta release)!

### Bug Fixes:

1. Submit a github issue - try to describe what happened so others can reproduce it too (config, code samples, expected vs. actual behaviour).

Note, that the sample code shall be minimal and if needed with publicly available data.

2. Try to fix it or recommend a solution...

We highly recommend to use test driven approach

\* convert your minimal code example to a unit/integration test with assert on expected results

\* start with debugging the issue... you can run just this particular test in your IDE and draft a fix

\* verify that your test case fails on the master branch and only passes with the fix applied

3. Submit a PR!

\_\*\*Note\*\*, even if you do not find the solution, sending a PR with a test covering the issue is a valid contribution and we can help you or finish it with you :]\_

### New Features:

1. Submit a github issue - describe what is the motivation of such feature (adding the use case or an example is helpful).

2. Let's discuss to determine the feature scope.

3. Submit a PR! (with updated docs and tests?).

---

## Guidelines

### Original code

All added or edited code shall be the own original work of the particular contributor.

If you use some third-party implementation, all such blocks/functions/modules shall be properly referred and if possible also agreed by code's author. For example - `This code is inpired from http://...`.

In case you adding new dependencies, make sure that they are compatible with the actual PyTorch Lightning license (ie. dependencies should be \_at least\_ as permissive as the PyTorch Lightning license).

### Coding Style

1. Use f-strings for output formation (except logging when we stay with lazy `logging.info("Hello %s!`, name);

2. Black code formatter is used using `pre-commit` hook.

### Documentation

We are using Sphinx with Napoleon extension.

Moreover we set Google style to follow with type convention.

- [Napoleon formatting with Google style](https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example\_google.html)

- [ReStructured Text (reST)](https://docs.pylonsproject.org/projects/docs-style-guide/)

- [Paragraph-level markup](https://www.sphinx-doc.org/en/1.5/markup/para.html)

See following short example of a sample function taking one position string and optional

```python

from typing import Optional

def my\_func(param\_a: int, param\_b: Optional[float] = None) -> str:

"""Sample function.

Args:

param\_a: first parameter

param\_b: second parameter

Return:

sum of both numbers

Example:

Sample doctest example...

>>> my\_func(1, 2)

3

.. note:: If you want to add something.

"""

p = param\_b if param\_b else 0

return str(param\_a + p)

```

When updating the docs make sure to build them first locally and visually inspect the html files (in the browser) for

formatting errors. In certain cases, a missing blank line or a wrong indent can lead to a broken layout.

Run these commands

```bash

pip install -r requirements/docs.txt

cd docs

make html

```

and open `docs/build/html/index.html` in your browser.

Notes:

- You need to have LaTeX installed for rendering math equations. You can for example install TeXLive by doing one of the following:

\* on Ubuntu (Linux) run `apt-get install texlive` or otherwise follow the instructions on the TeXLive website

\* use the [RTD docker image](https://hub.docker.com/r/readthedocs/build)

- with PL used class meta you need to use python 3.7 or higher

When you send a PR the continuous integration will run tests and build the docs. You can access a preview of the html pages in the

\_Artifacts\_ tab in CircleCI when you click on the task named \_ci/circleci: Build-Docs\_ at the bottom of the PR page.

### Testing

Testing your work locally will help you speed up the process since it allows you to focus on particular (failing) test-cases.

To setup a local development environment, install both local and test dependencies:

```bash

python -m pip install -r requirements/devel.txt

python -m pip install -r requirements/examples.txt

python -m pip install pre-commit

```

You can run the full test-case in your terminal via this bash script:

```bash

bash .run\_local\_tests.sh

```

Note: if your computer does not have multi-GPU nor TPU these tests are skipped.

For convenience, you can also use your own CircleCI building which will be triggered with each commit.

This is useful if you do not test against all required dependency versions.

To do so, login to [CircleCI](https://app.circleci.com/) and enable your forked project in the dashboard. It will just work after that.

### Pull Request

We welcome any useful contribution! For your convenience here's a recommended workflow:

0. Think about what you want to do - fix a bug, repair docs, etc.

1. Start your work locally (usually until you need our CI testing)

- create a branch and prepare your changes

- hint: do not work with your master directly, it may become complicated when you need to rebase

- hint: give your PR a good name! it will be useful later when you may work on multiple tasks/PRs

2. Create a "Draft PR" which is clearly marked, to let us know you don't need feedback yet.

3. When you feel ready for integrating your work, mark your PR "Ready for review".

4. Use tags in PR name for following cases:

- \*\*[blocked by #<number>]\*\* if you work is depending on others changes

- \*\*[wip]\*\* when you start to re-edit your work, mark it so no one will accidentally merge it in meantime

### Question & Answer

1. \*\*How can I help/contribute?\*\*

All help is very welcome - reporting bugs, solving issues and preparing bug fixes. To solve some issues you can start with label [good first issue](https://github.com/PyTorchLightning/pytorch-lightning/issues?q=is%3Aopen+is%3Aissue+label%3A%22good+first+issue%22) or chose something close to your domain with label [help wanted](https://github.com/PyTorchLightning/pytorch-lightning/issues?q=is%3Aopen+is%3Aissue+label%3A%22help+wanted%22). Before you start to implement anything check that the issue description that it is clear and self-assign the task to you (if it is not possible, just comment that you take it and we assign it to you...).

2. \*\*Is there a recommendation for branch names?\*\*

We do not rely on the name convention so far you are working with your own fork. Anyway it would be nice to follow this convention `<type>/<issue-id>\_<short-name>` where the types are: `bugfix`, `feature`, `docs`, `tests`, ...

3. \*\*How to rebase my PR?\*\*

We recommend creating a PR in separate branch other than `master`, especially if you plan submitting several changes and do not want to wait until the fist one is resolved (we can work on them in parallel). Update your master with upstream (assuming you have already set [upstream](https://help.github.com/en/github/collaborating-with-issues-and-pull-requests/configuring-a-remote-for-a-fork))

```bash

git fetch --all --prune

git checkout master

git merge upstream/master

```

checkout your feature branch

```bash

git checkout my-PR-branch

git rebase master

# follow git instructions to resolve conflists

git push -f

```