Warning This page is under construction

This document discusses the Continuous Integration (CI) system for PyTorch.

Currently PyTorch utilizes Github Action, CircleCI and Jenkins CI for various different CI build/test configurations. We will discuss these related contents in the following sections:

- CI System Overview
- CI Matrix
  - Basic Configurations
  - Other Variances
- Entrypoint for CI
  - Build System
  - Test System
- What is CI testing and When
  - CI workflow on PR
    - Using Github Labels
    - Using CIFlow
      - Architecture
      - User Guide
      - Implementation Details
  - CI workflow on master commits
  - Change CI workflow behavior on PR
  - Disabled tests in CI
    - How to disable a test
    - How to test the disabled test on CI
- Other Topics
  - CI Metrics
  - CI Internals
  - Binary Builds
  - Docker Builds
  - Other Related Repos
  - Which commit is used in CI for your PR?
  - CI failure tips on PR

CI System Overview

PyTorch CI system ensures that proper build/test process should pass on both PRs and master commits. There are several terminologies we would like to clarify:

- *CI job*: a CI job consist of a sequence of predefined steps, each of which will execute some specific script to build or test PyTorch. For example:
- *CI workflow*: a CI workflow consist of a collection of CI jobs that are depending on each other. For example:
- *CI run*: Refers to all the CI workflows triggered on a single commit (either on PR or on master). For example:

- *CI platform*: we currently have 3 CI platforms (Github Actions, CirlceCI, Jenkins).

PyTorch supports many different hardware architectures, operation system, accelerator GPU. Therefore, many different CI workflows run parallel on each commit to ensure that PyTorch can be built and run correctly in different environments and configurations. See CI Matrix section for more info.

For historic reasons we currently have 3 different CI platforms and there are dozens of CI jobs run on these 3 platforms. For each platform specific information please refer to the CI Internals section for more info.

CI Matrix

Currently there are 3 different categories of CI runs for a single commit: CI run (1) on PR, (2) on master commits, (3) on nightly commits. These CI workflow configurations changes from time to time (Please refer to the CI Internals section for more detail), but in general the rule of thumb for our CI runs are. 1. We consider the set of CI workflows run on master commits are the base line. 2. We run on PRs a subset of CI workflows comparing to the base line we run on master commits. 3. We mostly run binary CI workflows and smoke tests on nightly commits.

For more details on which CI workflow is being run on which category, please refer to the section: What is CI testing and When.

Each one of these triggers a specific subset of CI workflows defined in the CI matrix. Currently our CI matrix consists of some combination of the following basic configurations.

Basic Configurations

Note examples are based on the CI workflows at the time of writing.

1. Python versions (e.g. 3.6 - 3.9)
2. OS versions (e.g. Linux xenial/bionic, Windows 10, macos 10.13/10.15)
3. Compute Hardware (e.g. CPU, CUDA10/11, ROCM4)
4. Compiler (GCC 5.4/7/9, Clang5/7/9)

Obviously not every Cartesian product of these combination is being tested. Please refer to the PyTorch CI HUD for more information on all the combinations run currently.

Other Variances

Note variances in configurations can change more rapidly comparing to the basic configurations.

Other than the 4 basic configuration coordinates, we also have some special variances in configurations that we run. These variances are created to test some

specific features or to cover some specific test domains. For example:

1. *ASAN build*: to run address sanitizers.
2. *libtorch builds*: to target CPP APIs.
3. *coverage builds*: to report test coverages.
4. *pure torch build*: to test PyTorch built without Caffe2.
5. *ONNX build*: to test ONNX integration.
6. *XLA/Vulkan build*: to test XLA integration.
7. *Android/IOS build*: to test PyTorch run mobile platforms.

Entrypoint for CI

Build System

Test System

See "Running and writing tests" page for information.

What is CI testing and When

CI workflow on PR

Using Github Labels See "how to use GitHub labels" section on the "Running and writing tests" page for information.

Using CIFlow CIFlow is a flexible CI workflow dispatcher that's going to dispatch GitHub Actions CI workflows based on various PR contexts and user instructions. See the discussion on the RFC here.

Architecture

User Guide Note: CIFlow is automatically enabled for everyone (starting 10/1/2021), if you want to opt out, please follow the instructions in this issue https://github.com/pytorch/pytorch/issues/64124.

- Create PR as normal, and @pytorchbot will automatically handle the rest. It also generates a comment block **"CI Flow Status"**.
- If the PR author or anyone with `write` permission to the pytorch/pytorch repo wants to instrument @pytorchbot to run different configurations of the CI flow, people can **comment on the PR with the commands** like the following to trigger CIFlow.

```
# ciflow rerun, "ciflow/default" will always be added automatically
@pytorchbot ciflow rerun

# ciflow rerun with additional labels "-l <ciflow/label_name>", which is equivalent to addi
@pytorchbot ciflow rerun -l ciflow/scheduled -l ciflow/slow
```

Implementation Details The pytorchbot (which is defined here) will leverage multiple strategies to orchestrate how GitHub Action workflows are run or not run. The decision is made into two steps:

- PR author or @pytorchbot will add labels to the PR, e.g. `ciflow/default` (which is added automatically), or other labels.
- @pytorchbot will trigger an `unassign` event on the PR, which triggers all the workflows that are listening on `on.pull_request - [unassign]` events, and that event will have information of the `ciflow/*` labels, which is used to decide the condition.
  - See the generate_ci_workflows.py definition
  - See the sample ciflow_should_run's `if` condition.

CI workflow on master commits

Change CI workflow behavior on PR

Disabled tests in CI

Some PyTorch tests are currently disabled due to their flakiness, incompatibility with certain platforms, or other temporary brokenness. We have a system where GitHub issues titled "DISABLED test_a_name" disable desired tests in PyTorch CI until the issues are closed, e.g., #62970. If you are wondering what tests are currently disabled in CI, please check out disabled-tests.json, where these test cases are all gathered.

How to disable a test First, you should never disable a test if you're not sure what you're doing. Tests are important in validating PyTorch functionality, and ignoring test failures is not recommended as it can degrade user experience. When you are certain that disabling a test is the best option, for example, when it is flaky, make plans to fix the test so that it is not disabled indefinitely.

To disable a test, say, create an issue with the title `DISABLED test_case_name (test.ClassName)`. A real title example would look like: `DISABLED test_jit_cuda_extension (__main__.TestCppExtensionJIT)`. In the body of the issue, feel free to include any details and logs as you normally would with any issue. If you would like to skip the test for particular platforms, such as ROCm, please include a line (case insensitive) in the issue body like so: "Platforms: Mac, Windows." The available platforms to choose from are: mac/macos, windows, rocm, linux, and asan (whether the test is with ASAN).

How to test the disabled test on CI It is not easy to test these disabled tests with CI because, well, they're intentionally disabled. Previous alternatives were to either mimic the test environment locally (often not convenient) or to close the issue and re-enable the test in all of CI (risking breaking trunk CI).

After #62851 and #74981, PRs with key phrases like "fixes #55555" or "Close https://github.com/pytorch/pytorch/issues/62359" in their PR bodies or commit

messages will re-enable the tests disabled by the linked issues (in this example, #55555 and #62359). Including a key phrase in the PR body only works for tests triggered by pull request and does not work for push-triggered CI. Including a key phrase in the commit message will work for both pull and push triggered CI. The test will also be run if ANY of the commit messages in the PR contains a key phrase.

More accepted key phrases are defined by the GitHub docs.

Limitations: this feature only works for GitHub Actions CI.

Other Topics

CI Metrics

A compilation of dashboards and metrics relating to CI could be found in the PyTorch CI Metrics Dashboards wiki

CI Internals

Binary Builds

Docker Builds

See: https://github.com/pytorch/pytorch/wiki/Docker-image-build-on-CircleCI for more information.

Other Related Repos

pytorch/builder

pytorch/test-infra

pytorch/pytorch-ci-hud

Which commit is used in CI for your PR?

CI failure tips on PR