Validation of Out-of-Tree Accelerator Integration for PyTorch

TL;DR

With the rapid growth of the PyTorch ecosystem, the mechanism to integrate and test Out-Of-Tree accelerators is much more important than before. Establishing a quality monitoring and evaluation system for out-of-tree accelerators is crucial for the diverse development of PyTorch accelerators, and also good for the generalized design/impl of PyTorch core to accommodate multiple device backends.

This proposal aims to build up a general quality assurance, integration and test mechanism (such as installation verification, model validation, etc.) for out-of-tree accelerators, as well as a portal for these accelerators. Offering developers and users a unified platform for efficiently evaluating and utilizing various accelerators.

Motivation

The test cases for in-tree accelerators like CPU/CUDA are directly integrated within the PyTorch project, and all submissions must pass these tests before they can be merged. However, for out-of-tree accelerators (e.g., Intel HPU and Ascend NPU), since the code is not hosted within the PyTorch project and these accelerators are often less well-known, only a few developers familiar with them can fix issues when tests fail. This problem also arises for accelerators that are transitioning to in-tree status (e.g., Intel XPU).

However, the functional validation of these accelerators is essential. Currently, this work is done in the repositories of accelerator vendors, leading to inconsistent testing capabilities. When users encounter unexpected issues while using these accelerators, the lack of a unified testing standard negatively impacts the user experience with PyTorch and significantly affects the diverse development of accelerators.

Therefore, maintaining a set of testing standards in the community and deploying these tests in an independent repository can ensure basic functionality for these accelerators. This would also facilitate developers and users in selecting accelerators based on actual needs and provide a space for third-party accelerators to host relevant documentation, FAQs, and communication channels for vendor feedback.

Current Issues

1. **Version Compatibility**: Out-of-tree accelerator code is not hosted in the PyTorch project, making it difficult to keep in sync with the main PyTorch versions.
2. **Lack of Testing Capabilities**: The testing standards maintained by different accelerator vendors are not unified, leading to inconsistent testing capabilities and affecting accelerator stability.

3. **Limited Fixing Capabilities**: When test failures occur, only a few developers familiar with these backends can address them, making maintenance challenging.
4. **Compromised User Experience**: Users may encounter unexpected issues when using less common accelerators, negatively impacting the PyTorch user experience.
5. **Limited Diversity Development**: The absence of unified testing standards and documentation support restricts the diversity of the PyTorch accelerator ecosystem.

Advantages Offered By The Proposal

1. **Ensured Functional Stability**: Maintaining a standardized testing framework in the community ensures that the basic functionality of all accelerators is verified.
2. **Unified Documentation and Support**: Third-party accelerators can host documentation and FAQs in the community and establish feedback channels for communication with vendors.
3. **Promoting Accelerator Diversity**: With community support and standardized testing, the development and user experience of accelerator backends can be improved, fostering diversity in the ecosystem.

Approach

Testing Standards

Establishing testing standards for out-of-tree accelerators is a gradual process that requires collaborative efforts from various vendors. As a first step, we believe that building and deploying, as well as unit testing and model validation testing, are necessary. Among these, building and deploying, and unit testing are optional, as the build and unit test processes for closed-source accelerators may risk exposing implementation details. However, model validation testing is essential for all accelerators.

Compilation and Deployment (Optional)

Each backend needs to be compiled and validated using the accelerator version compatible with PyTorch. This step ensures that upstream modifications are properly implemented across all accelerators. Given the differences in drivers and development kits among third-party accelerators, vendors should provide methods for environment deployment and installation (including deployment scripts, Dockerfiles, or images) to verify the correctness of installation and deployment on standard operating systems (containers).

Unit Testing (Optional)

It is recommended that each accelerator run PyTorch's generic unit tests, which do not specify or differentiate between different accelerators. Additionally, custom unit test sets specific to the accelerator can be run to further ensure quality.

Model Validation Testing

Referencing PyTorch's end-to-end validation should be conducted using multiple models from Hugging Face Transformers, TIMM, and TorchBench. Not all third-party accelerators

may support all models, so it should be permissible for them to configure their setups to skip unsupported models. The results should indicate the execution success of these models for each accelerator, allowing users to assess their performance.

Guidelines and Result Presentation

Guidelines for the testing standards and the validation results for each accelerator need a dedicated space for storage. It is recommended to create an independent code repository for this purpose. Importantly, regardless of where the integration tests are run, the result presentation must include detailed information on the model validation process.

CI/CD Integration

Finding a way to ensure the quality of Out-of-Tree backend products while not blocking the PyTorch community CI system is both meaningful and challenging.

Event Triggered CI

When certain events occur in the official PyTorch repository (such as creating a new PR or merging a PR), the GitHub native repository dispatch mechanism can trigger the workflow of downstream public repositories. This process would compile the Out-of-Tree backend and run basic test cases to identify and record any upstream PRs that may cause breaks. After the verification is completed, the verification results will be written back to the original PR as a comment.

Advantages:

- It won't block the existing PyTorch CI system and won't affect the merging of community PRs.
- The PyTorch repository doesn't need to be aware of the workflows and devices of any Out-of-Tree backends.
- The Out-of-Tree repository can promptly detect how upstream changes affect it, promoting compatibility before a PR is merged by providing comments.

Disadvantages:

- A new workflow in PyTorch is needed to trigger the workflows of downstream public repositories.

Requests to PyTorch:

- A new public repository CI trigger workflow needs to be added to the PyTorch repository, which should be activated during specific events (like creating a new PR or merging a PR).

Periodic Running CI

A scheduled task would periodically fetch the main branch of PyTorch and the target branches of Out-of-Tree accelerators, testing compilation, running comprehensive unit tests, integration tests, etc. The results would be presented to users through a quality dashboard.

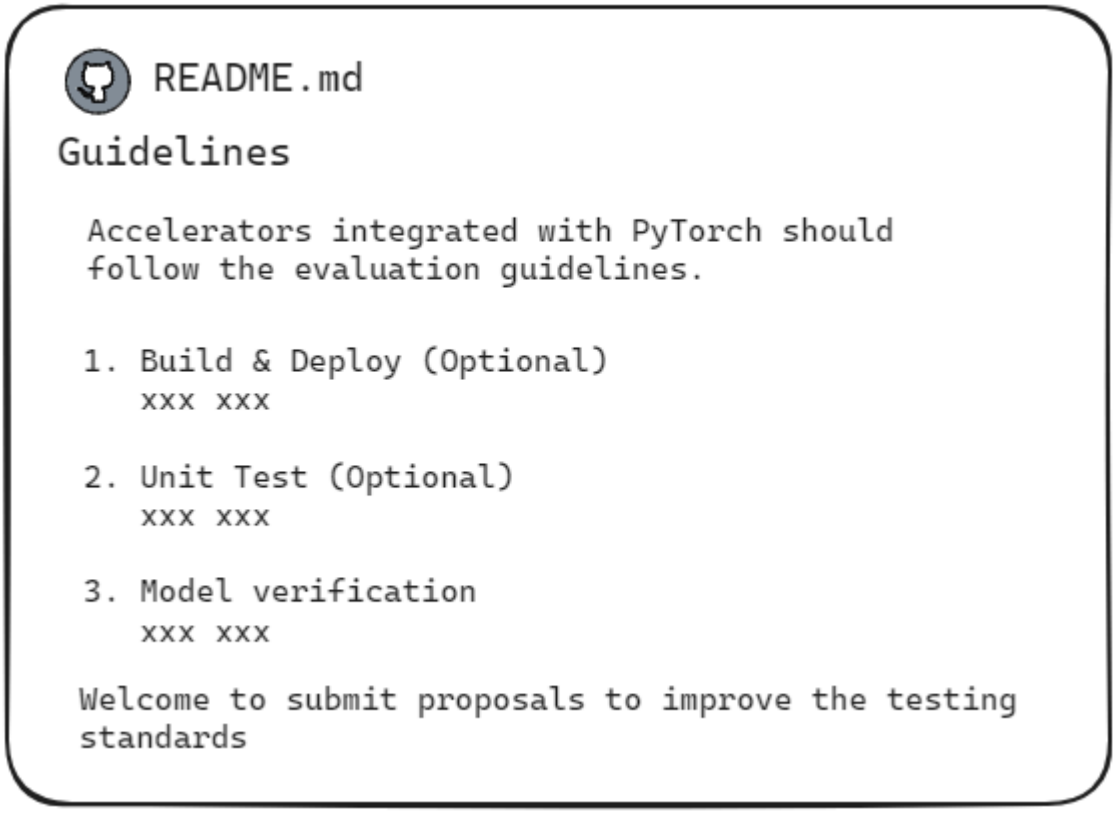
Requests to PyTorch:

- None.

Implementation Steps

Create a Shared Repository

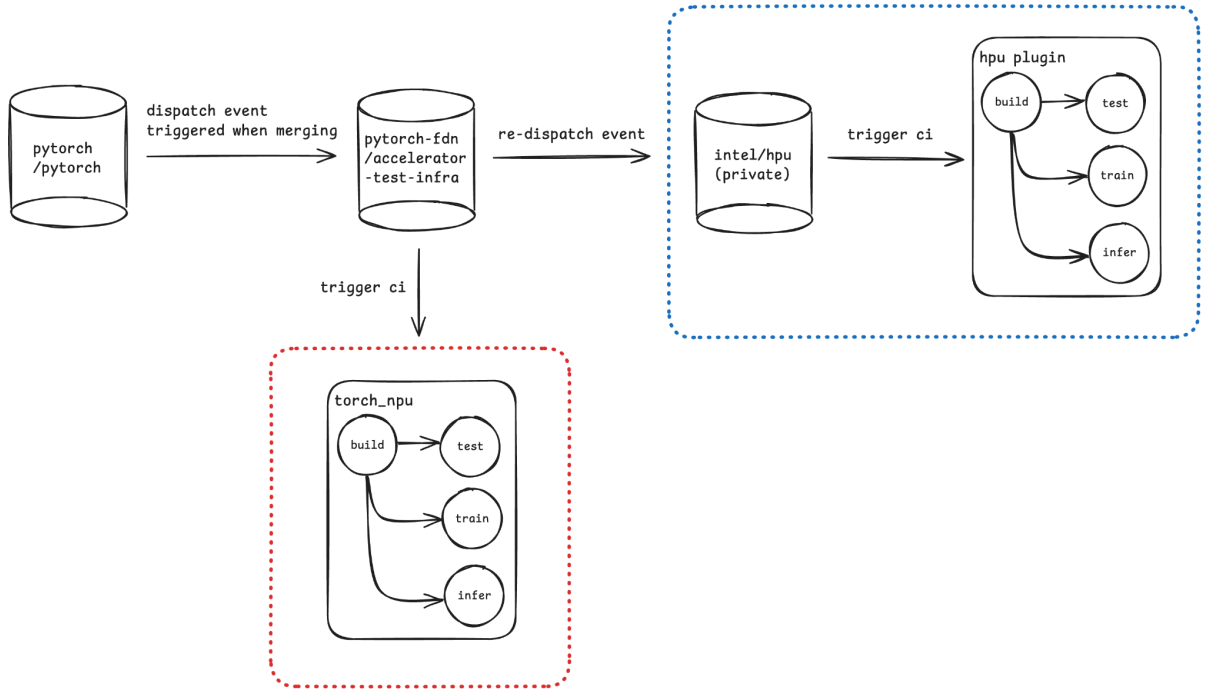
Host the CI of the current out-of-tree accelerators in a separate repository. We need a repository where participating out-of-tree accelerator vendors can jointly contribute and develop CI and testing standards (currently including Ascend NPU and Intel HPU). Therefore, a repository is needed to guide and showcase the results of the above. It is recommended to place it under the unified management of the pytorch or pytorch-fdn organization.



Workflow Trigger for PyTorch and the accelerator-test-infra Repository

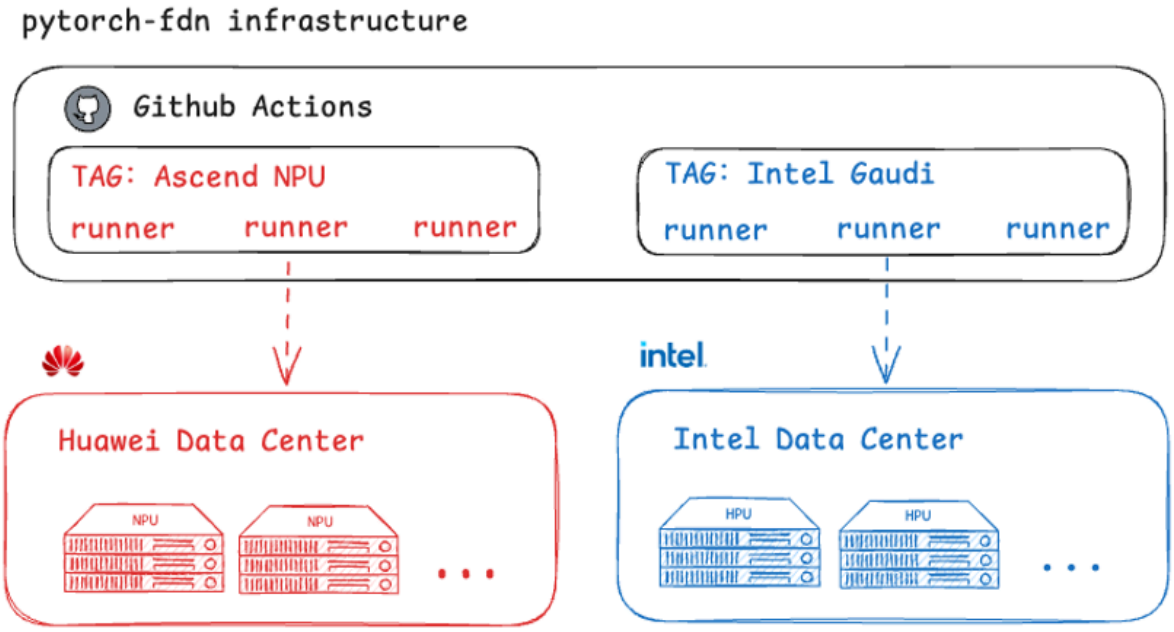
In the pytorch/pytorch repository, add the CI repository's PAT (Personal Access Token) to GitHub Secrets and create a new workflow. This workflow will listen for specific events and create a dispatch event, sending event information to the pytorch-fdn/accelerator-test-infra repository.
In the pytorch-fdn/accelerator-test-infra repository, create a new workflow to define actions

triggered by the dispatch event, which will trigger all out-of-tree CI workflows.
For out-of-tree devices with closed-source or self-hosted CI systems, the triggering mechanism must be implemented independently. This can be done by redispatching through the CI repository's workflow to the self-hosted CI system, or by directly registering the hardware in this repository and defining the task flow.
Devices integrated into the repository can independently decide which CI tasks to run. It is recommended to support tasks like compilation, TorchBench, and mainstream model validation.



Infrastructure Support (Optional)

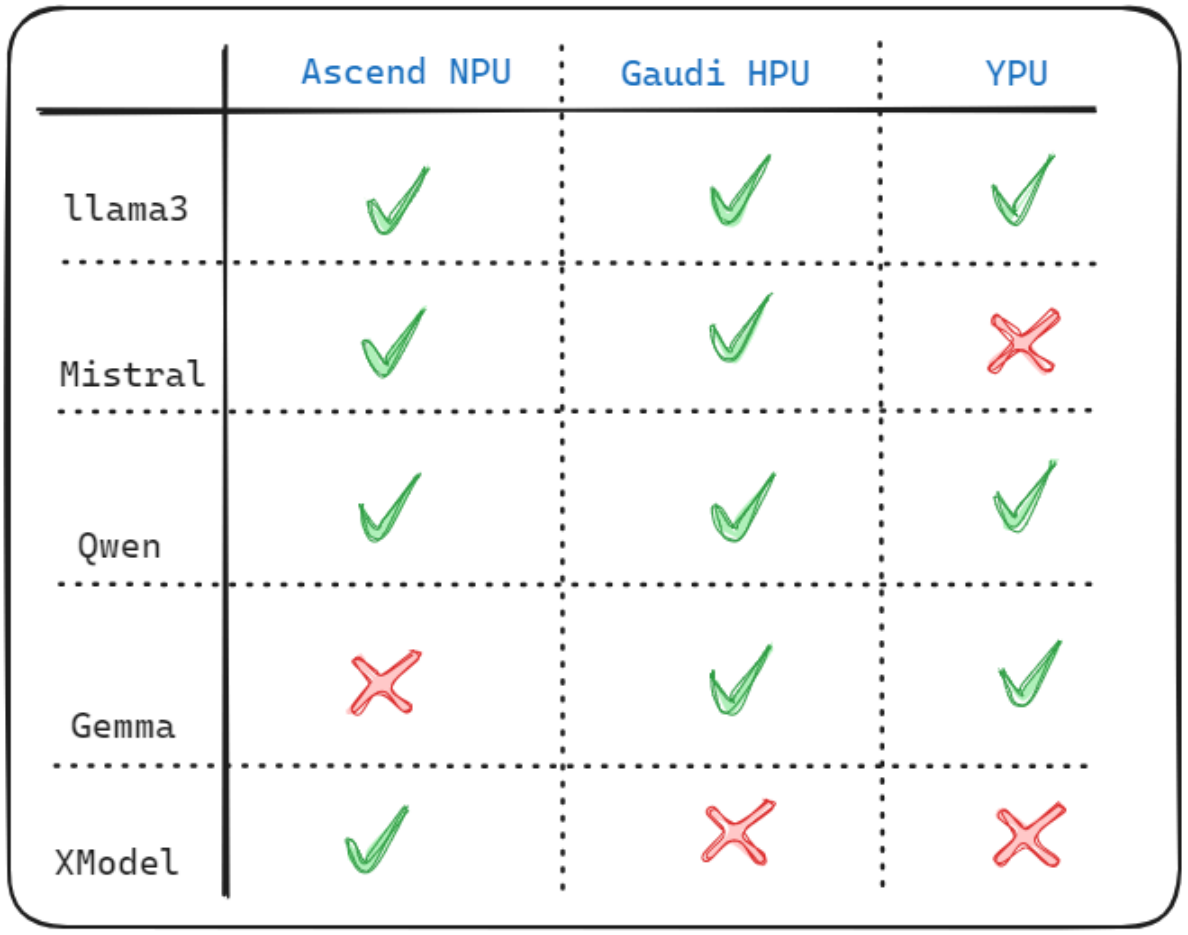
If a backend wishes to add and run CI in this repository, the accelerator vendor needs to connect the necessary hardware environment with the CI runner in the repository.



Due to differences in testing environments required by different accelerators, each accelerator vendor needs to provide their own CI server and add it to the GitHub Runner. Vendors are responsible for maintaining their servers to ensure availability.

Integration Testing

According to the testing standards, each accelerator should perform the standard tests either in this CI repository or in their own repository, and display the test results in the upstream repository. To allow users to view the detailed testing process, clicking on the test results should show the full verification process, ensuring that the testing aligns with the expected standards.



To facilitate local reproduction of the testing process, clicking on the accelerator's name should redirect to the relevant documentation for that accelerator, including environment setup and application deployment.
Each accelerator can customize the rules for triggering tests as needed. For example, tests may be triggered by each PR, nightly builds, or scheduled triggers. If a build or test case fails, the issue should be fixed immediately, and aside from large-scale features, it should be resolved within 72 hours.