Please make a copy of this doc and edit locally.

PyTorch Release Planning | Feature Submission Template

There are some examples from the past releases in Appendix.

Feature proposed:

Example: “`A torch.special module, analogous to SciPy’s special module.`”

torch.export() API, a generic way to extract full static graph from PyTorch

Point(s) of contact:

Yanan Cao (PyTorch) Avik Chaudhuri

Proposed release type:

Please review the criteria listed in the Appendix - Feature Classification.

	Stable
	Beta
✓	Prototype

Release Mode (pytorch/pytorch features only):

✓	In-tree
	Out-of-tree

If “out-of-tree”, please include the GH repo name

Description and value to the user (optional for prototype) :

Please include a link or pointer to user workflow if applicable.

torch.export() provides a sound tracing mechanism to capture a full graph from a PyTorch program based on new technologies provided by PT2.0.

Users can extract a clean representation (Export IR) of a PyTorch program in the form of a dataflow graph, consisting of mostly straight-line calls to PyTorch operators. Export IR can then be transformed, serialized, saved to file, transferred, loaded back for execution in an environment with or without Python.

Link to design doc, GitHub issues, past submissions, etc (optional for prototype):

What feedback adopters have provided:

Please list Meta and/or OSS users/teams that have tried the feature and provided feedback.

If that feedback motivated material changes (API, doc, etc..), a quick overview of the changes and the status (planned, in progress, implemented) would be helpful as well.

Note: This is most critical for features at the Prototype and Beta stage. At Stable stage, there should be more users (probably more than you can list) but it still might be useful to highlight key early users who could provide useful feedback on the feature.

Incremental requirements need to reach Stable (if applicable):

Example:

(optional for prototype) Serialization format changes, backward compatibility, breaking changes

Please describe whether you're extending persistent formats with this change - e.g. TorchScript format, operators which can appear in TorchScript programs, input data serialization, etc.

None

Plan for documentations / tutorials:

Based on the categories, we target the checklisted items for each feature. The following table shows the typical requirements for the features at different stages.

You'll notice that we have parity between stable and beta for these items. Depending on the feature we can back off on these but generally speaking we should be treating beta features in a similar manner and with a similar level of product thinking.

**** Can be internal, team members, and external users*

O - optional

Artifact	Requirement			Proposed Feature
	Prototype	Beta	Stable	
Doc Strings	✓	✓	✓	✓
Unit Tests	✓	✓	✓	✓
CI Coverage	✓	✓	✓	✓
Design Review / TL Signoff		✓	✓	✓
Recipe or Tutorial		✓	✓	✓
Mentioned in Released Blog(s)		✓	✓	✓

User Feedback (feature with User API surface)***		✓	✓	✓
Dogfooding: 1-2 early adopter teams (internal or external) have found the feature useful and their feedback has been incorporated		✓	✓	✓
API Stability			✓	✓
Top-line adoption metrics (internal or external), feedback from major companies and AI researchers has been incorporated			✓	✓

Please describe the plan for the artifacts you listed above. Also, include additional documentations you would write if given more time.

SELECT ONE OF THE FOLLOWING (change the color of the tick to black):

	Tutorial exists. Link: {{add link here}}
✓	Will submit a PR to pytorch/tutorials by 09/25/2023
	Will submit a PR to {{add library repo}}
	Tutorial is not needed. {{explain why}}

Add more details here:

Marketing/Blog Coverage

Are you requesting feature Inclusion in the release blogs? Reminder Beta/Stable requires release blogs.

✓	Yes
	No

Are you requesting other marketing assistance with this feature? E.g. supplementary blogs, social media amplification, etc.

Just release notes. Nothing else for now. Wait for PyTorch Conference Announcement

OS / Platform / Compute Coverage

Please list the platforms supported by the proposed feature. If the feature supports all the platforms, write “all”.

Goal of this section is to clearly share if this feature works in all PyTorch configurations or is it limited to only certain platforms/configurations (e.g. CPU only, GPU only, Linux only, etc...)

Supported Platforms	Limitations
all	

Example:

Supported Platforms	Limitations
Android	May work with a range of supported NDKs and Gradle, CPU only
iOS	CPU only

Testing Support (CI, test cases, etc..)

Please provide an overview of test coverage. This includes unit testing and integration testing, but if E2E validation testing has been done to show that the feature works for a certain set of use cases or models please mention that as well.

Also, include what additional tests you would implement if given more time.

torch.export() is covered in CI with unit tests and end-to-end real models.

Appendix

Examples

- [1.10] Conjugate View
- [1.10] PyTorch Mobile Tracing-based Mobile Interpreter
- [1.9] torch.linalg
- [1.9] Freezing API
- [1.8] FX
- [1.6] vmap

Feature Classification

The following classification was established at PyTorch 1.6 release. For details of what was changed then, please refer to PyTorch feature classification changes.

Stable

A stable feature means that the user value-add is or has been proven, the API isn't expected to change, the feature is performant and all documentation exists to support end user adoption.

Level of commitment: We expect to maintain these features long term and generally there should be no major performance limitations, gaps in documentation and we also expect to maintain backwards compatibility (although breaking changes can happen and notice will be given one release ahead of time).

Beta

The value-add, similar to a Stable feature, has been proven and the feature generally works and is documented. This feature is tagged as Beta because the API may change based on user feedback, because the performance needs to improve or because coverage across operators is not yet complete.

Level of commitment: We are committing to seeing the feature through to the Stable classification. We are however not committing to Backwards Compatibility. Users can depend on us providing a solution for problems in this area going forward, but the APIs and performance characteristics of this feature may change.

Prototype

The feature is not available as part of binary distributions like PyPI or Conda (except maybe behind run-time flags), but we would like to get high bandwidth partner feedback (see #Dogfooding) ahead of a real release in order to gauge utility and any changes we need to make to the UX.

To test these kinds of features we would, depending on the feature, recommend building from

master or using the nightly builds that are made available on pytorch.org. For each prototype feature, a pointer to draft docs or other instructions will be provided.

Level of commitment: We are committing to gathering high bandwidth feedback only. Based on this feedback and potential further engagement between community members, we as a community will decide if we want to upgrade the level of commitment or to fail fast. Additionally, while some of these features might be more speculative (e.g. new Frontend APIs), others have obvious utility (e.g. model optimization) but may be in a state where gathering feedback outside of high bandwidth channels is not practical, e.g. the feature may be in an earlier state, may be moving fast (PRs are landing too quickly to catch a major release) and/or generally active development is underway.