asteroid Documentation

Release 0.4.0rc1

Manuel Pariente et al.

Jan 05, 2021

Start here

Asteroid is a Pytorch-based audio source separation toolkit that enables fast experimentation on common datasets. It comes with a source code that supports a large range of datasets and architectures, and a set of recipes to reproduce some important papers.

CHAPTER 1

What is Asteroid?

Asteroid is a PyTorch-based audio source separation toolkit.

The main goals of Asteroid are:

- Gather a wider **community** around audio source separation by lowering the barriers to entry.
- **Promote reproducibility** by replicating important research papers.
- Automatize most engineering and **make way for research**.
- Simplify **model sharing** to reduce compute costs and carbon footprint.

So, how do we do that? We aim to provide

- PyTorch `Dataset` for **common datasets**.
- Ready-to-use state-of-the art source separation architectures in **native PyTorch**.
- **Configurable recipes** from data preparation to evaluation.
- **Pretrained models** for a wide variety of tasks and architectures.

1.1 Who is it for?

Asteroid has several target usage:

- Use asteroid in your own code, as a package.
- Use available recipes to build your own separation model.
- Use pretrained models to process your files.
- Hit the ground running with your research ideas!

1.2 Want to know more?

- Visit our webpage
- Read our paper
- Watch the presentation video
- **‘Check how we won the PyTorch Hackathon 2020 !<https://devpost.com/software/asteroid-the-pytorch-based-source-separation-toolkit>‘__**

CHAPTER 2

Installation

By following the instructions below, first install PyTorch and then Asteroid (using either pip/dev install). We recommend the development installation for users likely to modify the source code.

2.1 CUDA and PyTorch

Asteroid is based on PyTorch. To run Asteroid on GPU, you will need a CUDA-enabled PyTorch installation. Visit this site for the instructions: https://pytorch.org/get-started/locally/.

2.2 Pip

Asteroid is regularly updated on PyPI, install the latest stable version with:

```
pip install asteroid
```

2.3 Development installation

For development installation, you can fork/clone the GitHub repo and locally install it with pip:

```
git clone https://github.com/asteroid-team/asteroid
cd asteroid
pip install -e .
```

This is an editable install (`-e` flag), it means that source code changes (or branch switching) are automatically taken into account when importing asteroid.

You can also use `conda env create -f environment.yml` to create a Conda env directly.

CHAPTER 3

What is a recipe?

A recipe is a set of scripts that use Asteroid to build a source separation system. Each directory corresponds to a dataset and each subdirectory corresponds to a system build on this dataset. You can start by reading this recipe to get familiar with them.

3.1 How is it organized?

Most recipes are organized as follows. When you clone the repo, `data`, `exp` and `logs` won't be there yet, it's normal.

```
├── data/
├── exp/
├── logs/
├── local/
│   ├── convert_sphere2wav.sh
│   ├── prepare_data.sh
│   ├── conf.yml
│   └── preprocess_wham.py
├── utils/
│   ├── parse_options.sh
│   └── prepare_python_env.sh
├── run.sh
├── train.py
├── model.py
└── eval.py
```

A small graph might help.

3.2 How does it work?

Let's try to summarize how recipes work :

- There is a master file, `run.sh`, from which all the steps are ran (install dependencies, download data, create dataset, train a model evaluate it and so on..). This recipe style is borrowed from Kaldi and ESPnet.
  - You usually have to change some variables in the top of the file (comments are around it to help you) like data directory, python path etc..
  - This script is controlled by several arguments. Among them, `stage` controls from where do you start the script. You already generated the data? No need to do it again, set `stage=3`!
  - All steps until training are dataset-specific and the corresponding scripts are stored in `./local`
- The training and evaluation scripts are then called from `run.sh`
  - There is a script, `model.py`, where the model should be defined along with the `System` subclass used for training (if needed).
  - We wrap the model definition in one function (`make_model_and_optimizer`). The function receives a dictionary which is also saved in the experiment folder. This make checkpoint restoring easy without any additional constraints.
  - We also write a function to load the best model (`load_best_model`) after training. This is useful to load the model several time (evaluation, separation of new examples...).
- The arguments flow through bash/python/yaml in a specific way, which was designed by us and suits our use-cases until now:
  - The very first step is the `local/conf.yml` file where is a hierarchical configuration file,
  - **On the python side** : This file is parsed as a dictionary of dictionary in `training.py`. From this dict, we create an argument parser which can accept all the second-level keys from the dictionary (so second-level keys should be unique) as arguments and has the default values from the `conf.yml` file.
  - **On the bash side**: we also parse arguments from the command line (using `utils/parse_options.sh`). The arguments above the line `. utils/parse_options.sh` can be parsed, the rest are fixed. Most arguments will be passed to the training script. Others control the data preparation, GPU usage etc...
  - In light of all this the config file should have sensible default values that shouldn't be modified by hand much. The quickly configurable part of the recipe are added to `run.sh` (you want to experiment with the batch size, add an argument in and pass it to python. If you want it fixed, no need to put it in bash, the conf.yml file keeps it for you.) This makes it possible to directly identify the important parts of the experiment, without reading lots of lines of argparser or bash arguments.
- Some more notes :
  - After the first execution, you can go and change `stage` in `run.sh` to avoid redoing all the steps every-time.
  - To use GPUs for training, run `run.sh --id 0,1` where `0` and `1` are the GPUs you want to use, training should automatically take advantage of both GPUs.
  - By default, a random id is generated for each run, you can also add a `tag` to name the experiments how you want. For example `run.sh --tag with_cool_loss` will save all results to `exp/train_{arch_name}_with_cool_loss`. You'll also find the corresponding log file in `logs/train_{arch_name}_with_cool_loss.log`.
  - Model loading methods suppose that the model architecture is the same as when training was performed. Be careful when you change it.

Again, you have a doubt, a question, a suggestion or a request, open an issue or join the slack, we'll be happy to help you.

CHAPTER 4

Datasets and tasks

The following is a list of supported datasets, sorted by task. If you're more interested in the corresponding PyTorch `Dataset`, see *this page*

4.1 Speech separation

4.1.1 wsj0-2mix dataset

wsj0-2mix is a single channel speech separation dataset base on WSJ0. Three speaker extension (wsj0-3mix) is also considered here.

Reference

```
@article{Hershey_2016,
   title={Deep clustering: Discriminative embeddings for segmentation and separation},
   ISBN={9781479999880},
   url={http://dx.doi.org/10.1109/ICASSP.2016.7471631},
   DOI={10.1109/icassp.2016.7471631},
   journal={2016 IEEE International Conference on Acoustics, Speech and Signal␣
↪Processing (ICASSP)},
   publisher={IEEE},
   author={Hershey, John R. and Chen, Zhuo and Le Roux, Jonathan and Watanabe, Shinji}
↪,
   year={2016},
}
```

4.1.2 WHAM dataset

WHAM! is a noisy single-channel speech separation dataset based on WSJ0. It is a noisy extension of wsj0-2mix.

More info here.

References

```
@inproceedings{WHAMWichern2019,
  author={Gordon Wichern and Joe Antognini and Michael Flynn and Licheng Richard Zhu
↪and Emmett McQuinn and Dwight Crow and Ethan Manilow and Jonathan Le Roux},
  title={{WHAM!: extending speech separation to noisy environments}},
  year=2019,
  booktitle={Proc. Interspeech},
  pages={1368--1372},
  doi={10.21437/Interspeech.2019-2821},
  url={http://dx.doi.org/10.21437/Interspeech.2019-2821}
}
```

4.1.3 WHAMR dataset

WHAMR! is a noisy and reverberant single-channel speech separation dataset based on WSJ0. It is a reverberant extension of WHAM!.

Note that WHAMR! can synthesize binaural recordings, but we only consider the single channel for now.

More info here. **References**

```
@misc{maciejewski2019whamr,
    title={WHAMR!: Noisy and Reverberant Single-Channel Speech Separation},
    author={Matthew Maciejewski and Gordon Wichern and Emmett McQuinn and Jonathan Le
↪Roux},
    year={2019},
    eprint={1910.10279},
    archivePrefix={arXiv},
    primaryClass={cs.SD}
}
```

4.1.4 LibriMix dataset

The LibriMix dataset is an open source dataset derived from LibriSpeech dataset. It's meant as an alternative and complement to WHAM.

More info here.

References

```
@misc{cosentino2020librimix,
    title={LibriMix: An Open-Source Dataset for Generalizable Speech Separation},
    author={Joris Cosentino and Manuel Pariente and Samuele Cornell and Antoine
↪Deleforge and Emmanuel Vincent},
    year={2020},
    eprint={2005.11262},
    archivePrefix={arXiv},
    primaryClass={eess.AS}
}
```

4.1.5 Kinect-WSJ dataset

Kinect-WSJ is a reverberated, noisy version of the WSJ0-2MIX dataset. Microphones are placed on a linear array with spacing between the devices resembling that of Microsoft Kinect ™, the device used to record the CHiME-5 dataset.

This was done so that we could use the real ambient noise captured as part of CHiME-5 dataset. The room impulse responses (RIR) were simulated for a sampling rate of 16,000 Hz.

Requirements

- wsj_path : Path to precomputed wsj-2mix dataset. Should contain the folder 2speakers/wav16k/. If you don't have wsj_mix dataset, please create it using the scripts in egs/wsj0_mix
- chime_path : Path to chime-5 dataset. Should contain the folders train, dev and eval
- dihard_path : Path to dihard labels. Should contain `*.lab` files for the train and dev set

References Original repo

```
@inproceedings{sivasankaran2020,
  booktitle = {2020 28th {{European Signal Processing Conference}} ({{EUSIPCO}})},
  title={Analyzing the impact of speaker localization errors on speech separation for␣
↪automatic speech recognition},
  author={Sunit Sivasankaran and Emmanuel Vincent and Dominique Fohr},
  year={2021},
  month = Jan,
}
```

4.1.6 SMS_WSJ dataset

SMS_WSJ (stands for Spatialized Multi-Speaker Wall Street Journal) is a multichannel source separation dataset, based on WSJ0 and WSJ1.

All the information regarding the dataset can be found in this repo.

References If you use this dataset, please cite the corresponding paper as follows :

```
@Article{SmsWsj19,
  author    = {Drude, Lukas and Heitkaemper, Jens and Boeddeker, Christoph and Haeb-
↪Umbach, Reinhold},
  title     = {{SMS-WSJ}: Database, performance measures, and baseline recipe for␣
↪multi-channel source separation and recognition},
  journal   = {arXiv preprint arXiv:1910.13934},
  year      = {2019},
}
```

4.2 Speech enhancement

4.2.1 DNS Challenge's dataset

The Deep Noise Suppression (DNS) Challenge is a single-channel speech enhancement challenge organized by Microsoft, with a focus on real-time applications. More info can be found on the official page.

References The challenge paper, here.

```
@misc{DNSChallenge2020,
title={The INTERSPEECH 2020 Deep Noise Suppression Challenge: Datasets, Subjective␣
↪Speech Quality and Testing Framework},
author={Chandan K. A. Reddy and Ebrahim Beyrami and Harishchandra Dubey and Vishak␣
↪Gopal and Roger Cheng and Ross Cutler and Sergiy Matusevych and Robert Aichner and␣
↪Ashkan Aazami and Sebastian Braun and Puneet Rana and Sriram Srinivasan and␣
↪Johannes Gehrke}, year={2020},
```

(continues on next page)

(continued from previous page)

```
eprint={2001.08662},
}
```

The baseline paper, here.

```
@misc{xia2020weighted,
title={Weighted Speech Distortion Losses for Neural-network-based Real-time Speech␣
↪Enhancement},
author={Yangyang Xia and Sebastian Braun and Chandan K. A. Reddy and Harishchandra␣
↪Dubey and Ross Cutler and Ivan Tashev},
year={2020},
eprint={2001.10601},
}
```

4.3 Music source separation

4.3.1 MUSDB18 Dataset

The musdb18 is a dataset of 150 full lengths music tracks (~10h duration) of different genres along with their isolated drums, bass, vocals and others stems.

More info here.

4.3.2 DAMP-VSEP dataset

All the information regarding the dataset can be found in zenodo.

References If you use this dataset, please cite as follows :

```
@dataset{smule_inc_2019_3553059,
  author       = {Smule, Inc},
  title        = {{DAMP-VSEP: Smule Digital Archive of Mobile
                   Performances - Vocal Separation}},
  month        = oct,
  year         = 2019,
  publisher    = {Zenodo},
  version      = {1.0.1},
  doi          = {10.5281/zenodo.3553059},
  url          = {https://doi.org/10.5281/zenodo.3553059}
}
```

4.4 Environmental sound separation

4.4.1 FUSS dataset

The Free Universal Sound Separation (FUSS) dataset comprises audio mixtures of arbitrary sounds with source references for use in experiments on arbitrary sound separation.

All the information related to this dataset can be found in this repo.

References If you use this dataset, please cite the corresponding paper as follows:

```
@Article{Wisdom2020,
  author    = {Scott Wisdom and Hakan Erdogan and Daniel P. W. Ellis and Romain␣
↪Serizel and Nicolas Turpault and Eduardo Fonseca and Justin Salamon and Prem␣
↪Seetharaman and John R. Hershey},
  title     = {What's All the FUSS About Free Universal Sound Separation Data?},
  journal   = {in preparation},
  year      = {2020},
}
```

4.5 Audio-visual source separation

4.5.1 AVSpeech dataset

AVSpeech is an audio-visual speech separation dataset which was introduced by Google in this article Looking to Listen at the Cocktail Party: A Speaker-Independent Audio-Visual Model for Speech Separation.

More info here.

References

```
@article{Ephrat_2018,
   title={Looking to listen at the cocktail party},
   volume={37},
   url={http://dx.doi.org/10.1145/3197517.3201357},
   DOI={10.1145/3197517.3201357},
   journal={ACM Transactions on Graphics},
   publisher={Association for Computing Machinery (ACM)},
   author={Ephrat, Ariel and Mosseri, Inbar and Lang, Oran and Dekel, Tali and Wilson,
↪ Kevin and Hassidim, Avinatan and Freeman, William T. and Rubinstein, Michael},
   year={2018},
   pages={1-11}
}
```

4.6 Speaker extraction

CHAPTER 5

Training and Evaluation

Training and evaluation are the two essential parts of the recipes. For training, we offer a thin wrapper around PyTorchLightning that seamlessly enables distributed training, experiment logging and more, without sacrificing flexibility. For evaluation we released `pb_bss_eval` on PyPI, which is the evaluation part of pb_bss. All the credit goes to the original authors from the Paderborn University.

5.1 Training with PyTorchLightning

First, have a look here for an overview of PyTorchLightning. As you saw, the `LightningModule` is a central class of PyTorchLightning where a large part of the research-related logic lives. Instead of subclassing it everytime, we use `System`, a thin wrapper that separately gathers the essential parts of every deep learning project:

1. A model
2. Optimizer
3. Loss function
4. Train/val data

```
class System(pl.LightningModule):
        def __init__(self, model, optimizer, loss_func, train_loader, val_loader):
                ...

        def common_step(self, batch):
         """ common_step is the method that'll be called at both train/val time. """
                inputs, targets = batch
                est_targets = self(inputs)
                loss = self.loss_func(est_targets, targets)
                return loss
```

Only overwriting `common_step` will often be enough to obtain a desired behavior, while avoiding boilerplate code. Then, we can use the native PyTorchLightning `Trainer` to train the models.

5.2 Evaluation

Asteroid's function `compute_metrics` that calls `pb_bss_eval` is used to compute the following common source separation metrics:

- SDR / SIR / SAR
- STOI
- PESQ
- SI-SDR

CHAPTER 6

Pretrained models

Asteroid provides pretrained models through the Asteroid community in Zenodo. Have a look at the Zenodo page to choose which model you want to use.

Enjoy having pretrained models? **Please share your models** if you train some, we made it simple with the `asteroid-upload` CLI, check the next sections.

6.1 Using them

Loading a pretrained model is super simple!

```
from asteroid.models import ConvTasNet
model = ConvTasNet.from_pretrained('mpariente/ConvTasNet_WHAM!_sepclean')
```

Use the search page if you want to narrow your search.

You can also load it with Hub

```
from torch import hub
model = hub.load('mpariente/asteroid', 'conv_tasnet', 'mpariente/ConvTasNet_WHAM!_
↪sepclean')
```

6.2 Model caching

When using a `from_pretrained` method, the model is downloaded and cached. The cache directory is either the value in the `$ASTEROID_CACHE` environment variable, or `~/.cache/torch/asteroid`.

6.3 Share your models

At the end of each sharing-enabled recipe, all the necessary infos are gathered into a file, the only thing that's left to do is to run

```
asteroid-upload exp/your_exp_dir/publish_dir --uploader "Name Here"
```

Ok, not really. First you need to register to Zenodo (Sign in with GitHub: ok), create a token and use it with the `--token` option of the CLI, or by setting the `ACCESS_TOKEN` environment variable. If you plan to upload more models (and you should :innocent:), you can fill in your infos in `uploader_info.yml` at the root, like this.

```
uploader: Manuel Pariente
affiliation: INRIA
git_username: mpariente
token: TOKEN_HERE
```

6.4 Note about licenses

All Asteroid's pretrained models are shared under the Attribution-ShareAlike 3.0 (CC BY-SA 3.0) license. This means that models are released under the same license as the original training data. **If any non-commercial data is used during training (wsj0, WHAM's noises etc..), the models are non-commercial use only.** This is indicated in the bottom of the corresponding Zenodo page (ex: here).

CHAPTER 7

Command-line interface

7.1 Inference

7.1.1 asteroid-infer

Example

```
asteroid-infer "mpariente/ConvTasNet_WHAM!_sepclean" --files myaudio.wav --resample --
↪ola-window 8000 --ola-hop 4000
```

Reference

Command 'asteroid-infer –help' failed: [Errno 2] No such file or directory: 'asteroid-infer': 'asteroid-infer'

7.2 Publishing models

7.2.1 asteroid-upload

Reference

Command 'asteroid-upload –help' failed: [Errno 2] No such file or directory: 'asteroid-upload': 'asteroid-upload'

7.2.2 asteroid-register-sr

Reference

Command 'asteroid-register-sr –help' failed: [Errno 2] No such file or directory: 'asteroid-register-sr': 'asteroid-register-sr'

CHAPTER 8

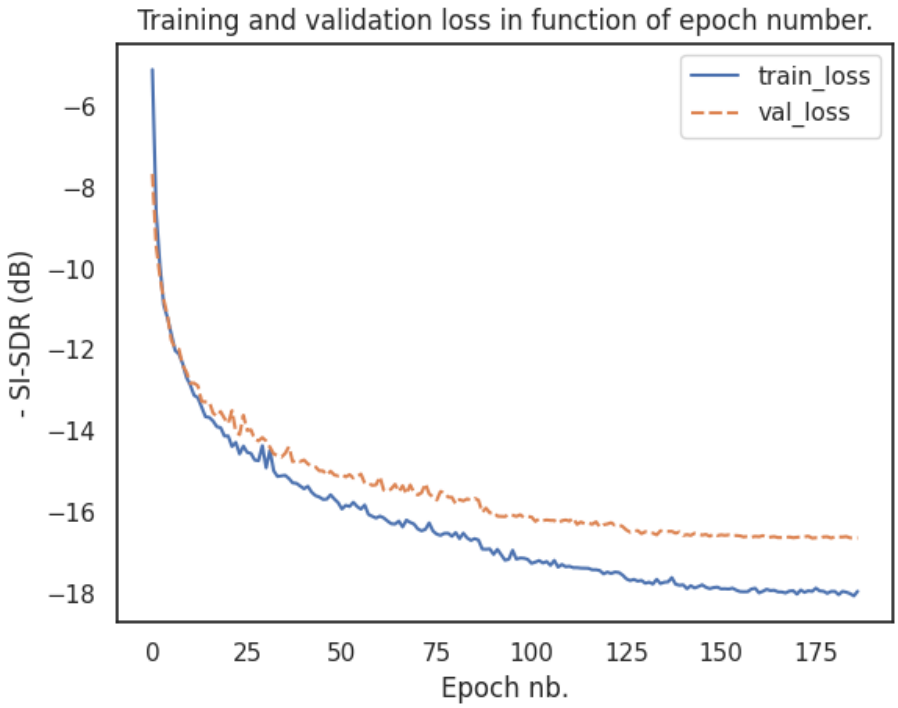
FAQ

8.1 My results are worse than the ones reported in the README, why?

There are few possibilities here:

1. Your data is wrong. We had this examples with wsj0-mix, WHAM etc.. where wv2 was used instead of wv1 to generate the data. This was fixed in #166. Chances are there is a pretrained model available for the given dataset, run the evaluation with it. If your results are different, it's a data problem. Refs: #164, #165 and #188.

2. You stopped training too early. We've seen this happen, specially with DPRNN. Be sure that your training/validation losses are completely flat at the end of training.



3. If it's not both, there is a real bug and we're happy you caught it! Please, open an issue with your torch/pytorch_lightning/asteroid versions to let us know.

8.2 How long does it take to train a model?

Need a log here.

8.3 Can I use the pretrained models for commercial purposes?

Not always. See the note on pretrained models Licenses *Note about licenses*

8.4 Separated audio is really bad, what is happening?

There are several possible cause to this, a common one is clipping.

1. When training with scale invariant losses (e.g. SI-SNR) the audio output can be unbounded. However, waveform values should be normalized to [-1, 1] range before saving, otherwise they will be clipped. See Clipping on Wikipedia and issue #250

2. As all supervised learning approaches, source separation can suffer from generalization error when evaluated on unseen data. If your model works well on data similar to your training data but doesn't work on real data, that's probably why. More about this on Wikipedia.

CHAPTER 9

PyTorch Datasets

This page lists the supported datasets and their corresponding PyTorch's `Dataset` class. If you're interested in the datasets more than in the code, see *this page*.

9.1 LibriMix

9.2 Wsj0mix

9.3 WHAM!

9.4 WHAMR!

9.5 SMS-WSJ

9.6 KinectWSJMix

9.7 DNSDataset

9.8 MUSDB18

9.9 DAMP-VSEP

9.10 FUSS

9.11 AVSpeech

CHAPTER 10

Filterbank API

10.1 Filterbank, Encoder and Decoder

10.2 Learnable filterbanks

10.2.1 Free

10.2.2 Analytic Free

10.2.3 Parameterized Sinc

10.3 Fixed filterbanks

10.3.1 STFT

10.3.2 MelGram

10.3.3 MPGT

10.4 Transforms

10.4.1 Griffin-Lim and MISI

10.4.2 Complex transforms

CHAPTER 11

DNN building blocks

CHAPTER 12

Models

12.1 Base classes

12.2 Ready-to-use models

12.3 Publishing models

CHAPTER 13

Losses & Metrics

13.1 Permutation invariant training (PIT) made easy

Asteroid supports regular Permutation Invariant Training (PIT), it's extension using Sinkhorn algorithm (SinkPIT) as well as Mixture Invariant Training (MixIT).

13.1.1 PIT

13.1.2 MixIT

13.1.3 SinkPIT

13.2 Available loss functions

`PITLossWrapper` supports three types of loss function. For "easy" losses, we implement the three types (pairwise point, single-source loss and multi-source loss). For others, we only implement the single-source loss which can be aggregated into both PIT and nonPIT training.

13.2.1 MSE

13.2.2 SDR

13.2.3 PMSQE

13.2.4 STOI

13.2.5 MultiScale Spectral Loss

13.2.6 Deep clustering (Affinity) loss

13.3 Computing metrics

CHAPTER 14

Lightning Wrapper

As explained in *Training and Evaluation*, Asteroid provides a thin wrapper on the top of PyTorchLightning for training your models.

CHAPTER 15

Optimizers & Schedulers

15.1 Optimizers

Asteroid relies on torch_optimizer and `torch` for optimizers. We provide a simple `get` method that retrieves optimizers from string, which makes it easy to specify optimizers from the command line.

Here is a list of supported optimizers, retrievable from string:

- AccSGD
- AdaBound
- AdaMod
- DiffGrad
- Lamb
- NovoGrad
- PID
- QHAdam
- QHM
- RAdam
- SGDW
- Yogi
- Ranger
- RangerQH
- RangerVA
- Adam
- RMSprop

- SGD
- Adadelta
- Adagrad
- Adamax
- AdamW
- ASG

15.2 Schedulers

Asteroid provides step-wise learning schedulers, integrable to `pytorch-lightning` via `System`.

CHAPTER 16

DSP Modules

CHAPTER 17

Utils

17.1 Parser utils

Asteroid has its own argument parser (built on `argparse`) that handles dict-like structure, created from a config YAML file.

17.2 Torch utils

17.3 Hub utils

17.4 Generic utils

CHAPTER 18

Asteroid High-Level Contribution Guide

Asteroid is a Pytorch-based audio source separation toolkit that enables fast experimentation on common datasets.

18.1 The Asteroid Contribution Process

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

Asteroid 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.
  - Join us on Slack and let us know you're interested in getting to know Asteroid. 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 datasets or architectures to Asteroid. Design discussion in these cases boils down mostly to, "Do we want this dataset/architecture?" Giving evidence for its utility, e.g., usage in peer reviewed papers, or existence in other frameworks, helps a bit when making this case.

  - Core changes and refactors can be quite difficult to coordinate, as the pace of development on Asteroid 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 and read the code for advice for working with Asteroid 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.
- **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.

18.2 Getting Started

18.2.1 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 Asteroid team and community frequently reviews new issues and comments where they think they can help. If you feel confident in your solution, go ahead and implement it.

18.2.2 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.

18.2.3 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. We can provide guidance that saves you time.

18.2.4 Adding Tutorials

Most our tutorials come from our team but we are very open to additional contributions. Have a notebook leveraging Asteroid? Open a PR to let us know!

18.2.5 Improving Documentation & Tutorials

We aim to produce high quality documentation and tutorials. On some 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.

18.2.6 Participating in online discussions

You can find active discussions happening on our slack workspace.

18.2.7 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.

18.2.8 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, additional eyes on pull requests is always appreciated.

18.2.9 Improving code readability

Improve code readability helps everyone. We plan to integrate `black`/DeepSource in the CI process, but readability issues can still persist and we'll welcome your corrections.

18.2.10 Adding test cases to make the codebase more robust

Additional test coverage is **always** appreciated.

18.2.11 Promoting Asteroid

Your use of Asteroid in your projects, research papers, write ups, blogs, or general discussions around the internet helps to raise awareness for Asteroid and our growing community. Please reach out to us for support.

18.2.12 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 an you feel an issue isn't categorized properly comment and let the team know.

18.3 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 in 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, are able to review subsequent changes and make a bugfix to the code. This naturally leads to a higher standard of contribution.

18.4 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 is 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. Difficulty of reviewing a PR scales nonlinearly with its size. You can try to split it up if possible, else 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 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 a hack is the right answer. But usually we will have to discuss it.
- **Do you want to touch a very core component?** In order 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 prior to 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.

18.5 Frequently asked questions

- **How can I contribute as a reviewer?** There is lots of value if community developer 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.

18.6 Attribution

This Contribution Guide is adapted from PyTorch's Contribution Guide available here.

CHAPTER 19

How to contribute

The general way to contribute to Asteroid is to fork the main repository on GitHub:

1. Fork the main repo and `git clone` it.
2. Make your changes, test them, commit them and push them to your fork.
3. You can open a pull request on GitHub when you're satisfied.

Things don't need to be perfect for PRs to be opened.

If you made changes to the source code, you'll want to try them out without installing asteroid everytime you change something. To do that, install asteroid in develop mode either with pip `pip install -e .[tests]` or with python `python setup.py develop`.

To avoid formatting roundtrips in PRs, Asteroid relies on *''black'* <https://github.com/psf/black>'_ and *''pre-commit-hooks'* <https://github.com/pre-commit/pre-commit-hooks>'_ to handle formatting for us. You'll need to install `requirements.txt` and install git hooks with `pre-commit install`.

Here is a summary:

```
### Install
git clone your_fork_url
cd asteroid
pip install -r requirements.txt
pip install -e .
pre-commit install  # To run black before commit

# Make your changes
# Test them locally
# Commit your changes
# Push your changes
# Open a PR!
```

19.1 Source code contributions

All contributions to the source code of asteroid should be documented and unit-tested. See here to run the tests with coverage reports. Docstrings follow the Google format, have a look at other docstrings in the codebase for examples. Examples in docstrings can be bery useful, don't hesitate to add some!

19.2 Writing new recipes.

Most new recipes should follow the standard format that is described here. We are not dogmatic about it, but another organization should be explained and motivated. We welcome any recipe on standard or new datasets, with standard or new architectures. You can even link a paper submission with a PR number if you'd like!

19.3 Improving the docs.

If you found a typo, think something could be more explicit etc... Improving the documentation is always welcome. The instructions to install dependencies and build the docs can be found here. Docstrings follow the Google format, have a look at other docstrings in the codebase for examples.

19.4 Coding style

We use pre-commit hooks to format the code using `black`. The code is checked for `black`- and `flake8`- compliance on every commit with GitHub actions. Remember, continuous integration is not here to be all green, be to help us see where to improve !

If you have any question, open an issue or join the slack, we'll be happy to help you.

CHAPTER 20

Indices and tables

- genindex
- modindex
- search