How to contribute to transformers?

Everyone is welcome to contribute, and we value everybody's contribution. Code is thus not the only way to help the community. Answering questions, helping others, reaching out and improving the documentations are immensely valuable to the community.

It also helps us if you spread the word: reference the library from blog posts on the awesome projects it made possible, shout out on Twitter every time it has helped you, or simply star the repo to say "thank you".

Whichever way you choose to contribute, please be mindful to respect our code of conduct.

You can contribute in so many ways!

There are 4 ways you can contribute to transformers:

- Fixing outstanding issues with the existing code;
- Implementing new models;
- Contributing to the examples or to the documentation;
- Submitting issues related to bugs or desired new features.

In particular there is a special Good First Issue listing. It will give you a list of open Issues that are open to anybody to work on. Just comment in the issue that you'd like to work on it. In that same listing you will also find some Issues with `Good Second Issue` label. These are typically slightly more complicated than the Issues with just `Good First Issue` label. But if you feel you know what you're doing, go for it.

All are equally valuable to the community.

Submitting a new issue or feature request

Do your best to follow these guidelines when submitting an issue or a feature request. It will make it easier for us to come back to you quickly and with good feedback.

Did you find a bug?

The 🤗 Transformers library is robust and reliable thanks to the users who notify us of the problems they encounter. So thank you for reporting an issue.

First, we would really appreciate it if you could **make sure the bug was not already reported** (use the search bar on Github under Issues).

Did not find it? :(So we can act quickly on it, please follow these steps:

- Include your **OS type and version**, the versions of **Python**, **PyTorch** and **Tensorflow** when applicable;
- A short, self-contained, code snippet that allows us to reproduce the bug in less than 30s;
- Provide the *full* traceback if an exception is raised.

To get the OS and software versions automatically, you can run the following command:

```
transformers-cli env
```

or from the root of the repository the following command:

```
python src/transformers/commands/transformers_cli.py env
```

Do you want to implement a new model?

Awesome! Please provide the following information:

- Short description of the model and link to the paper;
- Link to the implementation if it is open-source;
- Link to the model weights if they are available.

If you are willing to contribute the model yourself, let us know so we can best guide you.

We have added a **detailed guide and templates** to guide you in the process of adding a new model. You can find them in the `templates` folder.

Do you want a new feature (that is not a model)?

A world-class feature request addresses the following points:

1. Motivation first:

- Is it related to a problem/frustration with the library? If so, please explain why. Providing a code snippet that demonstrates the problem is best.
- Is it related to something you would need for a project? We'd love to hear about it!
- Is it something you worked on and think could benefit the community? Awesome! Tell us what problem it solved for you.

2. Write a *full paragraph* describing the feature;
3. Provide a **code snippet** that demonstrates its future use;
4. In case this is related to a paper, please attach a link;
5. Attach any additional information (drawings, screenshots, etc.) you think may help.

If your issue is well written we're already 80% of the way there by the time you post it.

We have added **templates** to guide you in the process of adding a new example script for training or testing the models in the library. You can find them in the `templates` folder.

Start contributing! (Pull Requests)

Before writing code, we strongly advise you to search through the existing PRs or issues to make sure that nobody is already working on the same thing. If you are unsure, it is always a good idea to open an issue to get some feedback.

You will need basic `git` proficiency to be able to contribute to 🤗 Transformers. `git` is not the easiest tool to use but it has the greatest manual. Type `git --help` in a shell and enjoy. If you prefer books, Pro Git is a very good reference.

Follow these steps to start contributing:

1. Fork the repository by clicking on the 'Fork' button on the repository's page. This creates a copy of the code under your GitHub user account.

2. Clone your fork to your local disk, and add the base repository as a remote:

```
$ git clone git@github.com:<your Github handle>/transformers.git
$ cd transformers
$ git remote add upstream https://github.com/huggingface/transformers.git
```

3. Create a new branch to hold your development changes:

```
$ git checkout -b a-descriptive-name-for-my-changes
```

Do not work on the `main` branch.

4. Set up a development environment by running the following command in a virtual environment:

   ```
   $ pip install -e ".[dev]"
   ```

   (If transformers was already installed in the virtual environment, remove it with `pip uninstall transformers` before reinstalling it in editable mode with the `-e` flag.)

   To run the full test suite, you might need the additional dependency on `datasets` which requires a separate source install:

   ```
   $ git clone https://github.com/huggingface/datasets
   $ cd datasets
   $ pip install -e .
   ```

   If you have already cloned that repo, you might need to `git pull` to get the most recent changes in the `datasets` library.

5. Develop the features on your branch.

   As you work on the features, you should make sure that the test suite passes. You should run the tests impacted by your changes like this:

   ```
   $ pytest tests/<TEST_TO_RUN>.py
   ```

   You can also run the full suite with the following command, but it takes a beefy machine to produce a result in a decent amount of time now that Transformers has grown a lot. Here is the command for it:

   ```
   $ make test
   ```

   For more information about tests, check out the [dedicated documentation](#)

   🤗 Transformers relies on `black` and `isort` to format its source code consistently. After you make changes, apply automatic style corrections and code verifications that can't be automated in one go with:

   ```
   $ make fixup
   ```

   This target is also optimized to only work with files modified by the PR you're working on.

   If you prefer to run the checks one after the other, the following command apply the style corrections:

   ```
   $ make style
   ```

   🤗 Transformers also uses `flake8` and a few custom scripts to check for coding mistakes. Quality control runs in CI, however you can also run the same checks with:

```
$ make quality
```

Finally we have a lot of scripts that check we didn't forget to update some files when adding a new model, that you can run with

```
$ make repo-consistency
```

To learn more about those checks and how to fix any issue with them, check out the documentation

If you're modifying documents under `docs/source`, make sure to validate that they can still be built. This check also runs in CI. To run a local check make sure you have installed the documentation builder requirements. First you will need to clone the repository containing our tools to build the documentation:

```
$ pip install git+https://github.com/huggingface/doc-builder
```

Then, make sure you have all the dependencies to be able to build the doc with:

```
$ pip install ".[docs]"
```

Finally run the following command from the root of the repository:

```
$ doc-builder build transformers docs/source/ --build_dir ~/tmp/test-build
```

This will build the documentation in the `~/tmp/test-build` folder where you can inspect the generated Markdown files with your favorite editor. You won't be able to see the final rendering on the website before your PR is merged, we are actively working on adding a tool for this.

Once you're happy with your changes, add changed files using `git add` and make a commit with `git commit` to record your changes locally:

```
$ git add modified_file.py
$ git commit
```

Please write good commit messages.

It is a good idea to sync your copy of the code with the original repository regularly. This way you can quickly account for changes:

```
$ git fetch upstream
$ git rebase upstream/main
```

Push the changes to your account using:

```
$ git push -u origin a-descriptive-name-for-my-changes
```

6. Once you are satisfied (**and the checklist below is happy too**), go to the webpage of your fork on GitHub. Click on 'Pull request' to send your changes to the project maintainers for review.

7. It's ok if maintainers ask you for changes. It happens to core contributors too! So everyone can see the changes in the Pull request, work in your local branch and push the changes to your fork. They will automatically appear in the pull request.

Checklist

1. The title of your pull request should be a summary of its contribution;
2. If your pull request addresses an issue, please mention the issue number in the pull request description to make sure they are linked (and people consulting the issue know you are working on it);
3. To indicate a work in progress please prefix the title with `[WIP]`. These are useful to avoid duplicated work, and to differentiate it from PRs ready to be merged;
4. Make sure existing tests pass;
5. Add high-coverage tests. No quality testing = no merge.
    - If you are adding a new model, make sure that you use `ModelTester.all_model_classes = (MyModel, MyModelWithLMHead,...)`, which triggers the common tests.
    - If you are adding new `@slow` tests, make sure they pass using `RUN_SLOW=1 python -m pytest tests/test_my_new_model.py`.
    - If you are adding a new tokenizer, write tests, and make sure `RUN_SLOW=1 python -m pytest tests/test_tokenization_{your_model_name}.py` passes. CircleCI does not run the slow tests, but github actions does every night!
6. All public methods must have informative docstrings that work nicely with sphinx. See `modeling_bert.py` for an example.
7. Due to the rapidly growing repository, it is important to make sure that no files that would significantly weigh down the repository are added. This includes images, videos and other non-text files. We prefer to leverage a hf.co hosted `dataset` like the ones hosted on `hf-internal-testing` in which to place these files and reference them by URL. We recommend putting them in the following dataset: huggingface/documentation-images. If an external contribution, feel free to add the images to your PR and ask a Hugging Face member to migrate your images to this dataset.

See more about the checks run on a pull request in our PR guide

Tests

An extensive test suite is included to test the library behavior and several examples. Library tests can be found in the tests folder and examples tests in the examples folder.

We like `pytest` and `pytest-xdist` because it's faster. From the root of the repository, here's how to run tests with `pytest` for the library:

```
$ python -m pytest -n auto --dist=loadfile -s -v ./tests/
```

and for the examples:

```
$ pip install -r examples/xxx/requirements.txt  # only needed the first time
$ python -m pytest -n auto --dist=loadfile -s -v ./examples/
```

In fact, that's how `make test` and `make test-examples` are implemented (sans the `pip install` line)!

You can specify a smaller set of tests in order to test only the feature you're working on.

By default, slow tests are skipped. Set the `RUN_SLOW` environment variable to `yes` to run them. This will download many gigabytes of models — make sure you have enough disk space and a good Internet connection, or a lot of patience!

```
$ RUN_SLOW=yes python -m pytest -n auto --dist=loadfile -s -v ./tests/
$ RUN_SLOW=yes python -m pytest -n auto --dist=loadfile -s -v ./examples/
```

Likewise, set the `RUN_CUSTOM_TOKENIZERS` environment variable to `yes` to run tests for custom tokenizers, which don't run by default either.

🤗 Transformers uses `pytest` as a test runner only. It doesn't use any `pytest` -specific features in the test suite itself.

This means `unittest` is fully supported. Here's how to run tests with `unittest` :

```
$ python -m unittest discover -s tests -t . -v
$ python -m unittest discover -s examples -t examples -v
```

Style guide

For documentation strings, 🤗 Transformers follows the google style. Check our documentation writing guide for more information.

This guide was heavily inspired by the awesome scikit-learn guide to contributing

Develop on Windows

On windows, you need to configure git to transform Windows `CRLF` line endings to Linux `LF` line endings:

`git config core.autocrlf input`

One way one can run the make command on Window is to pass by MSYS2:

1. Download MSYS2, we assume to have it installed in C:\msys64
2. Open the command line C:\msys64\msys2.exe (it should be available from the start menu)
3. Run in the shell: `pacman -Syu` and install make with `pacman -S make`
4. Add `C:\msys64\usr\bin` to your PATH environment variable.

You can now use `make` from any terminal (Powershell, cmd.exe, etc) 🎉

Syncing forked main with upstream (HuggingFace) main

To avoid pinging the upstream repository which adds reference notes to each upstream PR and sends unnecessary notifications to the developers involved in these PRs, when syncing the main branch of a forked repository, please, follow these steps:

1. When possible, avoid syncing with the upstream using a branch and PR on the forked repository. Instead merge directly into the forked main.
2. If a PR is absolutely necessary, use the following steps after checking out your branch:

```
$ git checkout -b your-branch-for-syncing
$ git pull --squash --no-commit upstream main
$ git commit -m '<your message without GitHub references>'
$ git push --set-upstream origin your-branch-for-syncing
```