# JAX/Flax Examples

This folder contains actively maintained examples of 🤗 Transformers using the JAX/Flax backend. Porting models and examples to JAX/Flax is an ongoing effort, and more will be added in the coming months. In particular, these examples are all designed to run fast on Cloud TPUs, and we include step-by-step guides to getting started with Cloud TPU.

*NOTE*: Currently, there is no "Trainer" abstraction for JAX/Flax -- all examples contain an explicit training loop.

The following table lists all of our examples on how to use 🤗 Transformers with the JAX/Flax backend:

- with information about the model and dataset used,
- whether or not they leverage the 🤗 Datasets library,
- links to **Colab notebooks** to walk through the scripts and run them easily.

| Task | Example model | Example dataset | 🤗 Datasets | Colab |
|---|---|---|---|---|
| `causal-language-modeling` | GPT2 | OSCAR | ✅ | Open in Colab |
| `masked-language-modeling` | RoBERTa | OSCAR | ✅ | Open in Colab |
| `text-classification` | BERT | GLUE | ✅ | Open in Colab |

## Intro: JAX and Flax

JAX is a numerical computation library that exposes a NumPy-like API with tracing capabilities. With JAX's `jit`, you can trace pure functions and compile them into efficient, fused accelerator code on both GPU and TPU. JAX supports additional transformations such as `grad` (for arbitrary gradients), `pmap` (for parallelizing computation on multiple devices), `remat` (for gradient checkpointing), `vmap` (automatic efficient vectorization), and `pjit` (for automatically sharded model parallelism). All JAX transformations compose arbitrarily with each other -- e.g., efficiently computing per-example gradients is simply `vmap(grad(f))`.

Flax builds on top of JAX with an ergonomic module abstraction using Python dataclasses that leads to concise and explicit code. Flax's "lifted" JAX transformations (e.g. `vmap`, `remat`) allow you to nest JAX transformation and modules in any way you wish. Flax is the most widely used JAX library, with 129 dependent projects as of May 2021. It is also the library underlying all of the official Cloud TPU JAX examples.

## Running on Cloud TPU

All of our JAX/Flax models are designed to run efficiently on Google Cloud TPUs. Here is a guide for running JAX on Google Cloud TPU.

Consider applying for the Google TPU Research Cloud project for free TPU compute.

Each example README contains more details on the specific model and training procedure.

## Running on single or multiple GPUs

All of our JAX/Flax examples also run efficiently on single and multiple GPUs. You can use the same instructions in the README to launch training on GPU. Distributed training is supported out-of-the box and scripts will use all the GPUs that are detected.

You should follow this guide for installing JAX on GPUs since the installation depends on your CUDA and CuDNN version.

## Supported models

Porting models from PyTorch to JAX/Flax is an ongoing effort. Feel free to reach out if you are interested in contributing a model in JAX/Flax -- we'll be adding a guide for porting models from PyTorch in the upcoming few weeks.

For a complete overview of models that are supported in JAX/Flax, please have a look at this table.

Over 3000 pretrained checkpoints are supported in JAX/Flax as of May 2021. Click here to see the full list on the 🤗 hub.

## Upload the trained/fine-tuned model to the Hub

All the example scripts support automatic upload of your final model to the Model Hub by adding a `--push_to_hub` argument. It will then create a repository with your username slash the name of the folder you are using as `output_dir`. For instance, `"sgugger/test-mrpc"` if your username is `sgugger` and you are working in the folder `~/tmp/test-mrpc`.

To specify a given repository name, use the `--hub_model_id` argument. You will need to specify the whole repository name (including your username), for instance `--hub_model_id sgugger/finetuned-bert-mrpc`. To upload to an organization you are a member of, just use the name of that organization instead of your username: `--hub_model_id huggingface/finetuned-bert-mrpc`.

A few notes on this integration:

- you will need to be logged in to the Hugging Face website locally for it to work, the easiest way to achieve this is to run `huggingface-cli login` and then type your username and password when prompted. You can also pass along your authentication token with the `--hub_token` argument.
- the `output_dir` you pick will either need to be a new folder or a local clone of the distant repository you are using.