


# CodeParrot

```
from torch import nn
from transformers import Model

class Transformer(nn.Module):
    def __init__(self, config):
        super(Transformer, self).__init__()
        self.config = config
        self.model = Model(config)

    def forward(self, inputs):
        return self.model(inputs)
```

## What is this about?

This is an open-source effort to train and evaluate code generation models. CodeParrot  is a GPT-2 model trained from scratch on Python code. The highlights of this project are:

- initialize and train a GPT-2 language model from scratch for code generation
- train a custom tokenizer adapted for Python code
- clean and deduplicate a large (>100GB) dataset with `datasets`
- train with `accelerate` on multiple GPUs using data parallelism and mixed precision
- continuously push checkpoints to the hub with `huggingface_hub`
- stream the dataset with `datasets` during training to avoid disk bottlenecks
- apply the `code_eval` metric in `datasets` to evaluate on [OpenAI's HumanEval benchmark](#)

## Installation

To install the dependencies simply run the following command:

```
pip install -r requirements.txt
```

To reproduce the results you can follow the scripts in the following sections. Note that we don't always show all possible arguments to the scripts. To get the full list of arguments with descriptions you can run the following command on any script:

```
python scripts/some_script.py --help
```

Before you run any of the scripts make sure you are logged in and can push to the hub:

```
huggingface-cli login
```

Additionally, sure you have git-lfs installed. You can find instructions for how to install it [here](#).

## Dataset

The source of the dataset is the GitHub dump available on Google's [BigQuery](#). The database was queried for all Python files with less than 1MB in size resulting in a 180GB dataset with over 20M files. The dataset is available on the Hugging Face Hub [here](#).

## Preprocessing

The raw dataset contains many duplicates. We deduplicated and filtered the dataset using the heuristics proposed in OpenAI's Codex [paper](#):

- exact deduplication using each file's hash
- filtering files with max line length > 1000
- filtering files with mean line length > 100
- fraction of alphanumeric characters < 0.25
- containing the word "auto-generated" or similar in the first 5 lines

The script to process the full dataset can be found in `scripts/preprocessing.py`. Executing the script on 16 vCPUs takes roughly 3h and removes 70% of the original dataset. The cleaned [train](#) and [validation](#) splits are also available on the Hub if you want to skip this step or use the data for another project.

To execute the preprocessing run the following command:

```
python scripts/preprocessing.py \  
--dataset_name lvwerra/codeparrot \  
--output_dir codeparrot-clean
```

During preprocessing the dataset is downloaded and stored locally as well as caches of the computations. Make sure you have more than 500GB free disk space to execute it.

## Tokenizer

Before training a new model for code we create a new tokenizer that is efficient at code tokenization. To train the tokenizer you can run the following command:

```
python scripts/bpe_training.py \  
--base_tokenizer gpt2 \  
--dataset_name lvwerra/codeparrot-clean-train
```

*Note:* We originally trained the tokenizer on the unprocessed train split of the dataset `transformersbook/codeparrot-train`.

## Training

The models are randomly initialized and trained from scratch. To initialize a new model you can run:

```
python scripts/initialize_model.py \  
--config_name gpt2-large \  
--tokenizer_name lvwerra/codeparrot \  
--model_name codeparrot \  
--push_to_hub True
```

This will initialize a new model with the architecture and configuration of `gpt2-large` and use the tokenizer to appropriately size the input embeddings. Finally, the initialized model is pushed to the hub.

Now that the dataset, tokenizer, and model are ready we can start training the model. The main training script is built with `accelerate` to scale across a wide range of platforms and infrastructure scales. We train two models with [110M](#) and [1.5B](#) parameters for 25-30B tokens on a 16xA100 (40GB) machine which takes 1 day and 1 week, respectively.

First you need to configure `accelerate` and login to Weights & Biases:

```
accelerate config
wandb login
```

Note that during the `accelerate` configuration we enabled FP16. Then to train the large model you can run

```
python scripts/codeparrot_training.py
```

If you want to train the small model you need to make some modifications:

```
accelerate launch scripts/codeparrot_training.py \
--model_ckpt lvwerra/codeparrot-small \
--train_batch_size 12 \
--valid_batch_size 12 \
--learning_rate 5e-4 \
--num_warmup_steps 2000 \
--gradient_accumulation 1 \
--gradient_checkpointing False \
--max_train_steps 150000 \
--save_checkpoint_steps 15000
```

Recall that you can see the full set of possible options with descriptions (for all scripts) by running:

```
python scripts/codeparrot_training.py --help
```

Instead of streaming the dataset from the hub you can also stream it from disk. This can be helpful for long training runs where the connection can be interrupted sometimes. To stream locally you simply need to clone the datasets and replace the dataset name with their path. In this example we store the data in a folder called `data` :

```
git lfs install
mkdir data
git -C "./data" clone https://huggingface.co/datasets/lvwerra/codeparrot-clean-train
git -C "./data" clone https://huggingface.co/datasets/lvwerra/codeparrot-clean-valid
```

And then pass the paths to the datasets when we run the training script:

```
accelerate launch scripts/codeparrot_training.py \
--model_ckpt lvwerra/codeparrot-small \
--dataset_name_train ./data/codeparrot-clean-train \
```

```
--dataset_name_valid ./data/codeparrot-clean-valid \  
--train_batch_size 12 \  
--valid_batch_size 12 \  
--learning_rate 5e-4 \  
--num_warmup_steps 2000 \  
--gradient_accumulation 1 \  
--gradient_checkpointing False \  
--max_train_steps 150000 \  
--save_checkpoint_steps 15000
```

## Evaluation

For evaluating the language modeling loss on the validation set or any other dataset you can use the following command:

```
python scripts/validation_loss.py \  
--model_ckpt lvwerra/codeparrot \  
--dataset_name lvwerra/codeparrot-clean-valid
```

In addition we evaluate the model on OpenAI's *HumanEval* benchmark. You can run the evaluation with the following command:

```
python scripts/human_eval.py --model_ckpt lvwerra/codeparrot \  
--do_sample True \  
--temperature 0.2 \  
--top_p 0.95 \  
--n_samples=200 \  
--HF_ALLOW_CODE_EVAL="0"
```

The results as well as reference values are shown in the following table:

Model	pass@1	pass@10	pass@100
CodeParrot 🦜 (110M)	3.80%	6.57%	12.78%
CodeParrot 🦜 (1.5B)	3.58%	8.03%	14.96%
Codex (25M)	3.21%	7.1%	12.89%
Codex (85M)	8.22%	12.81%	22.40%
Codex (300M)	13.17%	20.37%	36.27%
Codex (12B)	28.81%	46.81%	72.31%
GPT-neo (125M)	0.75%	1.88%	2.97%
GPT-neo (1.5B)	4.79%	7.47%	16.30%
GPT-neo (2.7B)	6.41%	11.27%	21.37%
GPT-J (6B)	11.62%	15.74%	27.74%

---

The numbers were obtained by sampling with  $\mathbf{T} = [0.2, 0.6, 0.8]$  and picking the best value for each metric. Both CodeParrot 🦜 models are still underfitted and longer training would likely improve the performance.

## Demo

Give the model a shot yourself! There are two demos to interact with CodeParrot 🦜:

- [Code generation](#)
- [Code highlighting](#)

## Further Resources

A detailed description of the project can be found in the chapter "Training Transformers from Scratch" in the upcoming O'Reilly book [Natural Language Processing with Transformers](#).

This example was provided by [Leandro von Werra](#).