**Trainer**

The [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) class provides an API for feature-complete training in PyTorch, and it supports distributed training on multiple GPUs/TPUs, mixed precision for [NVIDIA GPUs](https://nvidia.github.io/apex/), [AMD GPUs](https://rocm.docs.amd.com/en/latest/rocm.html), and [torch.amp](https://pytorch.org/docs/stable/amp.html) for PyTorch. [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) goes hand-in-hand with the [TrainingArguments](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments) class, which offers a wide range of options to customize how a model is trained. Together, these two classes provide a complete training API.

[Seq2SeqTrainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Seq2SeqTrainer) and [Seq2SeqTrainingArguments](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Seq2SeqTrainingArguments) inherit from the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) and TrainingArgument classes and they’re adapted for training models for sequence-to-sequence tasks such as summarization or translation.

The [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) class is optimized for 🤗 Transformers models and can have surprising behaviors when used with other models. When using it with your own model, make sure:

* your model always return tuples or subclasses of [ModelOutput](https://huggingface.co/docs/transformers/main/en/main_classes/output#transformers.utils.ModelOutput)
* your model can compute the loss if a labels argument is provided and that loss is returned as the first element of the tuple (if your model returns tuples)
* your model can accept multiple label arguments (use label\_names in [TrainingArguments](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments) to indicate their name to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) but none of them should be named "label"

**Trainer**

classtransformers.**Trainer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L236)

( model: typing.Union[transformers.modeling\_utils.PreTrainedModel, torch.nn.modules.module.Module] = Noneargs: TrainingArguments = Nonedata\_collator: typing.Optional[DataCollator] = Nonetrain\_dataset: typing.Optional[torch.utils.data.dataset.Dataset] = Noneeval\_dataset: typing.Union[torch.utils.data.dataset.Dataset, typing.Dict[str, torch.utils.data.dataset.Dataset], NoneType] = Nonetokenizer: typing.Optional[transformers.tokenization\_utils\_base.PreTrainedTokenizerBase] = Nonemodel\_init: typing.Union[typing.Callable[[], transformers.modeling\_utils.PreTrainedModel], NoneType] = Nonecompute\_metrics: typing.Union[typing.Callable[[transformers.trainer\_utils.EvalPrediction], typing.Dict], NoneType] = Nonecallbacks: typing.Optional[typing.List[transformers.trainer\_callback.TrainerCallback]] = Noneoptimizers: typing.Tuple[torch.optim.optimizer.Optimizer, torch.optim.lr\_scheduler.LambdaLR] = (None, None)preprocess\_logits\_for\_metrics: typing.Union[typing.Callable[[torch.Tensor, torch.Tensor], torch.Tensor], NoneType] = None )

Expand 11 parameters

**Parameters**

* **model** ([PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel) or torch.nn.Module, *optional*) — The model to train, evaluate or use for predictions. If not provided, a model\_init must be passed.

[Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) is optimized to work with the [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel) provided by the library. You can still use your own models defined as torch.nn.Module as long as they work the same way as the 🤗 Transformers models.

* **args** ([TrainingArguments](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments), *optional*) — The arguments to tweak for training. Will default to a basic instance of [TrainingArguments](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.TrainingArguments) with the output\_dir set to a directory named *tmp\_trainer* in the current directory if not provided.
* **data\_collator** (DataCollator, *optional*) — The function to use to form a batch from a list of elements of train\_dataset or eval\_dataset. Will default to [default\_data\_collator()](https://huggingface.co/docs/transformers/main/en/main_classes/data_collator#transformers.default_data_collator) if no tokenizer is provided, an instance of [DataCollatorWithPadding](https://huggingface.co/docs/transformers/main/en/main_classes/data_collator#transformers.DataCollatorWithPadding) otherwise.
* **train\_dataset** (torch.utils.data.Dataset or torch.utils.data.IterableDataset, *optional*) — The dataset to use for training. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed.

Note that if it’s a torch.utils.data.IterableDataset with some randomization and you are training in a distributed fashion, your iterable dataset should either use a internal attribute generator that is a torch.Generator for the randomization that must be identical on all processes (and the Trainer will manually set the seed of this generator at each epoch) or have a set\_epoch() method that internally sets the seed of the RNGs used.

* **eval\_dataset** (Union[torch.utils.data.Dataset, Dict[str, torch.utils.data.Dataset]), *optional*) — The dataset to use for evaluation. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. If it is a dictionary, it will evaluate on each dataset prepending the dictionary key to the metric name.
* **tokenizer** ([PreTrainedTokenizerBase](https://huggingface.co/docs/transformers/main/en/internal/tokenization_utils#transformers.PreTrainedTokenizerBase), *optional*) — The tokenizer used to preprocess the data. If provided, will be used to automatically pad the inputs to the maximum length when batching inputs, and it will be saved along the model to make it easier to rerun an interrupted training or reuse the fine-tuned model.
* **model\_init** (Callable[[], PreTrainedModel], *optional*) — A function that instantiates the model to be used. If provided, each call to [train()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.train) will start from a new instance of the model as given by this function.

The function may have zero argument, or a single one containing the optuna/Ray Tune/SigOpt trial object, to be able to choose different architectures according to hyper parameters (such as layer count, sizes of inner layers, dropout probabilities etc).

* **compute\_metrics** (Callable[[EvalPrediction], Dict], *optional*) — The function that will be used to compute metrics at evaluation. Must take a [EvalPrediction](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.EvalPrediction) and return a dictionary string to metric values.
* **callbacks** (List of [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback), *optional*) — A list of callbacks to customize the training loop. Will add those to the list of default callbacks detailed in [here](https://huggingface.co/docs/transformers/main/main_classes/callback).

If you want to remove one of the default callbacks used, use the [Trainer.remove\_callback()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.remove_callback) method.

* **optimizers** (Tuple[torch.optim.Optimizer, torch.optim.lr\_scheduler.LambdaLR], *optional*, defaults to (None, None)) — A tuple containing the optimizer and the scheduler to use. Will default to an instance of [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) on your model and a scheduler given by [get\_linear\_schedule\_with\_warmup()](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.get_linear_schedule_with_warmup) controlled by args.
* **preprocess\_logits\_for\_metrics** (Callable[[torch.Tensor, torch.Tensor], torch.Tensor], *optional*) — A function that preprocess the logits right before caching them at each evaluation step. Must take two tensors, the logits and the labels, and return the logits once processed as desired. The modifications made by this function will be reflected in the predictions received by compute\_metrics.

Note that the labels (second parameter) will be None if the dataset does not have them.

Trainer is a simple but feature-complete training and eval loop for PyTorch, optimized for 🤗 Transformers.

Important attributes:

* **model** — Always points to the core model. If using a transformers model, it will be a [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel) subclass.
* **model\_wrapped** — Always points to the most external model in case one or more other modules wrap the original model. This is the model that should be used for the forward pass. For example, under DeepSpeed, the inner model is wrapped in DeepSpeed and then again in torch.nn.DistributedDataParallel. If the inner model hasn’t been wrapped, then self.model\_wrapped is the same as self.model.
* **is\_model\_parallel** — Whether or not a model has been switched to a model parallel mode (different from data parallelism, this means some of the model layers are split on different GPUs).
* **place\_model\_on\_device** — Whether or not to automatically place the model on the device - it will be set to False if model parallel or deepspeed is used, or if the default TrainingArguments.place\_model\_on\_device is overridden to return False .
* **is\_in\_train** — Whether or not a model is currently running train (e.g. when evaluate is called while in train)

add\_callback

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L654)

( callback )

**Parameters**

* **callback** (type or [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback)) — A [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback) class or an instance of a [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback). In the first case, will instantiate a member of that class.

Add a callback to the current list of [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback).

autocast\_smart\_context\_manager

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2699)

( cache\_enabled: typing.Optional[bool] = True )

A helper wrapper that creates an appropriate context manager for autocast while feeding it the desired arguments, depending on the situation.

compute\_loss

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2750)

( modelinputsreturn\_outputs = False )

How the loss is computed by Trainer. By default, all models return the loss in the first element.

Subclass and override for custom behavior.

compute\_loss\_context\_manager

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2693)

( )

A helper wrapper to group together context managers.

create\_model\_card

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3521)

( language: typing.Optional[str] = Nonelicense: typing.Optional[str] = Nonetags: typing.Union[str, typing.List[str], NoneType] = Nonemodel\_name: typing.Optional[str] = Nonefinetuned\_from: typing.Optional[str] = Nonetasks: typing.Union[str, typing.List[str], NoneType] = Nonedataset\_tags: typing.Union[str, typing.List[str], NoneType] = Nonedataset: typing.Union[str, typing.List[str], NoneType] = Nonedataset\_args: typing.Union[str, typing.List[str], NoneType] = None )

**Parameters**

* **language** (str, *optional*) — The language of the model (if applicable)
* **license** (str, *optional*) — The license of the model. Will default to the license of the pretrained model used, if the original model given to the Trainer comes from a repo on the Hub.
* **tags** (str or List[str], *optional*) — Some tags to be included in the metadata of the model card.
* **model\_name** (str, *optional*) — The name of the model.
* **finetuned\_from** (str, *optional*) — The name of the model used to fine-tune this one (if applicable). Will default to the name of the repo of the original model given to the Trainer (if it comes from the Hub).
* **tasks** (str or List[str], *optional*) — One or several task identifiers, to be included in the metadata of the model card.
* **dataset\_tags** (str or List[str], *optional*) — One or several dataset tags, to be included in the metadata of the model card.
* **dataset** (str or List[str], *optional*) — One or several dataset identifiers, to be included in the metadata of the model card.
* **dataset\_args** (str or List[str], *optional*) — One or several dataset arguments, to be included in the metadata of the model card.

Creates a draft of a model card using the information available to the Trainer.

create\_optimizer

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L929)

( )

Setup the optimizer.

We provide a reasonable default that works well. If you want to use something else, you can pass a tuple in the Trainer’s init through optimizers, or subclass and override this method in a subclass.

create\_optimizer\_and\_scheduler

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L902)

( num\_training\_steps: int )

Setup the optimizer and the learning rate scheduler.

We provide a reasonable default that works well. If you want to use something else, you can pass a tuple in the Trainer’s init through optimizers, or subclass and override this method (or create\_optimizer and/or create\_scheduler) in a subclass.

create\_scheduler

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L1109)

( num\_training\_steps: intoptimizer: Optimizer = None )

**Parameters**

* **num\_training\_steps** (int) — The number of training steps to do.

Setup the scheduler. The optimizer of the trainer must have been set up either before this method is called or passed as an argument.

evaluate

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2988)

( eval\_dataset: typing.Union[torch.utils.data.dataset.Dataset, typing.Dict[str, torch.utils.data.dataset.Dataset], NoneType] = Noneignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'eval' )

**Parameters**

* **eval\_dataset** (Union[Dataset, Dict[str, Dataset]), *optional*) — Pass a dataset if you wish to override self.eval\_dataset. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. If it is a dictionary, it will evaluate on each dataset, prepending the dictionary key to the metric name. Datasets must implement the \_\_len\_\_ method.

If you pass a dictionary with names of datasets as keys and datasets as values, evaluate will run separate evaluations on each dataset. This can be useful to monitor how training affects other datasets or simply to get a more fine-grained evaluation. When used with load\_best\_model\_at\_end, make sure metric\_for\_best\_model references exactly one of the datasets. If you, for example, pass in {"data1": data1, "data2": data2} for two datasets data1 and data2, you could specify metric\_for\_best\_model="eval\_data1\_loss" for using the loss on data1 and metric\_for\_best\_model="eval\_data1\_loss" for the loss on data2.

* **ignore\_keys** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.
* **metric\_key\_prefix** (str, *optional*, defaults to "eval") — An optional prefix to be used as the metrics key prefix. For example the metrics “bleu” will be named “eval\_bleu” if the prefix is “eval” (default)

Run evaluation and returns metrics.

The calling script will be responsible for providing a method to compute metrics, as they are task-dependent (pass it to the init compute\_metrics argument).

You can also subclass and override this method to inject custom behavior.

evaluation\_loop

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3148)

( dataloader: DataLoaderdescription: strprediction\_loss\_only: typing.Optional[bool] = Noneignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'eval' )

Prediction/evaluation loop, shared by Trainer.evaluate() and Trainer.predict().

Works both with or without labels.

floating\_point\_ops

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3486)

( inputs: typing.Dict[str, typing.Union[torch.Tensor, typing.Any]] ) **→** int

**Parameters**

* **inputs** (Dict[str, Union[torch.Tensor, Any]]) — The inputs and targets of the model.

**Returns**

int

The number of floating-point operations.

For models that inherit from [PreTrainedModel](https://huggingface.co/docs/transformers/main/en/main_classes/model#transformers.PreTrainedModel), uses that method to compute the number of floating point operations for every backward + forward pass. If using another model, either implement such a method in the model or subclass and override this method.

get\_decay\_parameter\_names

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L918)

( model )

Get all parameter names that weight decay will be applied to

Note that some models implement their own layernorm instead of calling nn.LayerNorm, weight decay could still apply to those modules since this function only filter out instance of nn.LayerNorm

get\_eval\_dataloader

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L834)

( eval\_dataset: typing.Optional[torch.utils.data.dataset.Dataset] = None )

**Parameters**

* **eval\_dataset** (torch.utils.data.Dataset, *optional*) — If provided, will override self.eval\_dataset. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. It must implement \_\_len\_\_.

Returns the evaluation ~torch.utils.data.DataLoader.

Subclass and override this method if you want to inject some custom behavior.

get\_optimizer\_cls\_and\_kwargs

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L977)

( args: TrainingArguments )

**Parameters**

* **args** (transformers.training\_args.TrainingArguments) — The training arguments for the training session.

Returns the optimizer class and optimizer parameters based on the training arguments.

get\_test\_dataloader

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L869)

( test\_dataset: Dataset )

**Parameters**

* **test\_dataset** (torch.utils.data.Dataset, *optional*) — The test dataset to use. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. It must implement \_\_len\_\_.

Returns the test ~torch.utils.data.DataLoader.

Subclass and override this method if you want to inject some custom behavior.

get\_train\_dataloader

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L778)

( )

Returns the training ~torch.utils.data.DataLoader.

Will use no sampler if train\_dataset does not implement \_\_len\_\_, a random sampler (adapted to distributed training if necessary) otherwise.

Subclass and override this method if you want to inject some custom behavior.

hyperparameter\_search

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2561)

( hp\_space: typing.Union[typing.Callable[[ForwardRef('optuna.Trial')], typing.Dict[str, float]], NoneType] = Nonecompute\_objective: typing.Union[typing.Callable[[typing.Dict[str, float]], float], NoneType] = Nonen\_trials: int = 20direction: typing.Union[str, typing.List[str]] = 'minimize'backend: typing.Union[ForwardRef('str'), transformers.trainer\_utils.HPSearchBackend, NoneType] = Nonehp\_name: typing.Union[typing.Callable[[ForwardRef('optuna.Trial')], str], NoneType] = None\*\*kwargs ) **→** [trainer\_utils.BestRun or List[trainer\_utils.BestRun]]

Expand 7 parameters

**Parameters**

* **hp\_space** (Callable[["optuna.Trial"], Dict[str, float]], *optional*) — A function that defines the hyperparameter search space. Will default to default\_hp\_space\_optuna() or default\_hp\_space\_ray() or default\_hp\_space\_sigopt() depending on your backend.
* **compute\_objective** (Callable[[Dict[str, float]], float], *optional*) — A function computing the objective to minimize or maximize from the metrics returned by the evaluate method. Will default to default\_compute\_objective().
* **n\_trials** (int, *optional*, defaults to 100) — The number of trial runs to test.
* **direction** (str or List[str], *optional*, defaults to "minimize") — If it’s single objective optimization, direction is str, can be "minimize" or "maximize", you should pick "minimize" when optimizing the validation loss, "maximize" when optimizing one or several metrics. If it’s multi objectives optimization, direction is List[str], can be List of "minimize" and "maximize", you should pick "minimize" when optimizing the validation loss, "maximize" when optimizing one or several metrics.
* **backend** (str or ~training\_utils.HPSearchBackend, *optional*) — The backend to use for hyperparameter search. Will default to optuna or Ray Tune or SigOpt, depending on which one is installed. If all are installed, will default to optuna.
* **hp\_name** (Callable[["optuna.Trial"], str]], *optional*) — A function that defines the trial/run name. Will default to None.
* **kwargs** (Dict[str, Any], *optional*) — Additional keyword arguments passed along to optuna.create\_study or ray.tune.run. For more information see:
  + the documentation of [optuna.create\_study](https://optuna.readthedocs.io/en/stable/reference/generated/optuna.study.create_study.html)
  + the documentation of [tune.run](https://docs.ray.io/en/latest/tune/api_docs/execution.html#tune-run)
  + the documentation of [sigopt](https://app.sigopt.com/docs/endpoints/experiments/create)

**Returns**

**[**trainer\_utils.BestRun**or**List[trainer\_utils.BestRun]**]**

All the information about the best run or best runs for multi-objective optimization. Experiment summary can be found in run\_summary attribute for Ray backend.

Launch an hyperparameter search using optuna or Ray Tune or SigOpt. The optimized quantity is determined by compute\_objective, which defaults to a function returning the evaluation loss when no metric is provided, the sum of all metrics otherwise.

To use this method, you need to have provided a model\_init when initializing your [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer): we need to reinitialize the model at each new run. This is incompatible with the optimizers argument, so you need to subclass [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) and override the method [create\_optimizer\_and\_scheduler()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.create_optimizer_and_scheduler) for custom optimizer/scheduler.

init\_hf\_repo

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3504)

( )

Initializes a git repo in self.args.hub\_model\_id.

is\_local\_process\_zero

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2787)

( )

Whether or not this process is the local (e.g., on one machine if training in a distributed fashion on several machines) main process.

is\_world\_process\_zero

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2794)

( )

Whether or not this process is the global main process (when training in a distributed fashion on several machines, this is only going to be True for one process).

log

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2640)

( logs: typing.Dict[str, float] )

**Parameters**

* **logs** (Dict[str, float]) — The values to log.

Log logs on the various objects watching training.

Subclass and override this method to inject custom behavior.

log\_metrics

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_pt_utils.py#L905)

( splitmetrics )

**Parameters**

* **split** (str) — Mode/split name: one of train, eval, test
* **metrics** (Dict[str, float]) — The metrics returned from train/evaluate/predictmetrics: metrics dict

Log metrics in a specially formatted way

Under distributed environment this is done only for a process with rank 0.

Notes on memory reports:

In order to get memory usage report you need to install psutil. You can do that with pip install psutil.

Now when this method is run, you will see a report that will include: :

Copied

init\_mem\_cpu\_alloc\_delta = 1301MB

init\_mem\_cpu\_peaked\_delta = 154MB

init\_mem\_gpu\_alloc\_delta = 230MB

init\_mem\_gpu\_peaked\_delta = 0MB

train\_mem\_cpu\_alloc\_delta = 1345MB

train\_mem\_cpu\_peaked\_delta = 0MB

train\_mem\_gpu\_alloc\_delta = 693MB

train\_mem\_gpu\_peaked\_delta = 7MB

**Understanding the reports:**

* the first segment, e.g., train\_\_, tells you which stage the metrics are for. Reports starting with init\_ will be added to the first stage that gets run. So that if only evaluation is run, the memory usage for the \_\_init\_\_ will be reported along with the eval\_ metrics.
* the third segment, is either cpu or gpu, tells you whether it’s the general RAM or the gpu0 memory metric.
* \*\_alloc\_delta - is the difference in the used/allocated memory counter between the end and the start of the stage - it can be negative if a function released more memory than it allocated.
* \*\_peaked\_delta - is any extra memory that was consumed and then freed - relative to the current allocated memory counter - it is never negative. When you look at the metrics of any stage you add up alloc\_delta + peaked\_delta and you know how much memory was needed to complete that stage.

The reporting happens only for process of rank 0 and gpu 0 (if there is a gpu). Typically this is enough since the main process does the bulk of work, but it could be not quite so if model parallel is used and then other GPUs may use a different amount of gpu memory. This is also not the same under DataParallel where gpu0 may require much more memory than the rest since it stores the gradient and optimizer states for all participating GPUS. Perhaps in the future these reports will evolve to measure those too.

The CPU RAM metric measures RSS (Resident Set Size) includes both the memory which is unique to the process and the memory shared with other processes. It is important to note that it does not include swapped out memory, so the reports could be imprecise.

The CPU peak memory is measured using a sampling thread. Due to python’s GIL it may miss some of the peak memory if that thread didn’t get a chance to run when the highest memory was used. Therefore this report can be less than reality. Using tracemalloc would have reported the exact peak memory, but it doesn’t report memory allocations outside of python. So if some C++ CUDA extension allocated its own memory it won’t be reported. And therefore it was dropped in favor of the memory sampling approach, which reads the current process memory usage.

The GPU allocated and peak memory reporting is done with torch.cuda.memory\_allocated() and torch.cuda.max\_memory\_allocated(). This metric reports only “deltas” for pytorch-specific allocations, as torch.cuda memory management system doesn’t track any memory allocated outside of pytorch. For example, the very first cuda call typically loads CUDA kernels, which may take from 0.5 to 2GB of GPU memory.

Note that this tracker doesn’t account for memory allocations outside of [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)’s \_\_init\_\_, train, evaluate and predict calls.

Because evaluation calls may happen during train, we can’t handle nested invocations because torch.cuda.max\_memory\_allocated is a single counter, so if it gets reset by a nested eval call, train’s tracker will report incorrect info. If this [pytorch issue](https://github.com/pytorch/pytorch/issues/16266) gets resolved it will be possible to change this class to be re-entrant. Until then we will only track the outer level of train, evaluate and predict methods. Which means that if eval is called during train, it’s the latter that will account for its memory usage and that of the former.

This also means that if any other tool that is used along the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) calls torch.cuda.reset\_peak\_memory\_stats, the gpu peak memory stats could be invalid. And the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) will disrupt the normal behavior of any such tools that rely on calling torch.cuda.reset\_peak\_memory\_stats themselves.

For best performance you may want to consider turning the memory profiling off for production runs.

metrics\_format

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_pt_utils.py#L879)

( metrics: typing.Dict[str, float] ) **→** metrics (Dict[str, float])

**Parameters**

* **metrics** (Dict[str, float]) — The metrics returned from train/evaluate/predict

**Returns**

**metrics (**Dict[str, float]**)**

The reformatted metrics

Reformat Trainer metrics values to a human-readable format

num\_examples

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L1128)

( dataloader: DataLoader )

Helper to get number of samples in a ~torch.utils.data.DataLoader by accessing its dataset. When dataloader.dataset does not exist or has no length, estimates as best it can

num\_tokens

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L1142)

( train\_dl: DataLoadermax\_steps: typing.Optional[int] = None )

Helper to get number of tokens in a ~torch.utils.data.DataLoader by enumerating dataloader.

pop\_callback

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L665)

( callback ) **→** [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback)

**Parameters**

* **callback** (type or [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback)) — A [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback) class or an instance of a [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback). In the first case, will pop the first member of that class found in the list of callbacks.

**Returns**

[TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback)

The callback removed, if found.

Remove a callback from the current list of [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback) and returns it.

If the callback is not found, returns None (and no error is raised).

predict

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3086)

( test\_dataset: Datasetignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'test' )

**Parameters**

* **test\_dataset** (Dataset) — Dataset to run the predictions on. If it is an datasets.Dataset, columns not accepted by the model.forward() method are automatically removed. Has to implement the method \_\_len\_\_
* **ignore\_keys** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.
* **metric\_key\_prefix** (str, *optional*, defaults to "test") — An optional prefix to be used as the metrics key prefix. For example the metrics “bleu” will be named “test\_bleu” if the prefix is “test” (default)

Run prediction and returns predictions and potential metrics.

Depending on the dataset and your use case, your test dataset may contain labels. In that case, this method will also return metrics, like in evaluate().

If your predictions or labels have different sequence length (for instance because you’re doing dynamic padding in a token classification task) the predictions will be padded (on the right) to allow for concatenation into one array. The padding index is -100.

Returns: *NamedTuple* A namedtuple with the following keys:

* predictions (np.ndarray): The predictions on test\_dataset.
* label\_ids (np.ndarray, *optional*): The labels (if the dataset contained some).
* metrics (Dict[str, float], *optional*): The potential dictionary of metrics (if the dataset contained labels).

prediction\_loop

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3702)

( dataloader: DataLoaderdescription: strprediction\_loss\_only: typing.Optional[bool] = Noneignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'eval' )

Prediction/evaluation loop, shared by Trainer.evaluate() and Trainer.predict().

Works both with or without labels.

prediction\_step

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3381)

( model: Moduleinputs: typing.Dict[str, typing.Union[torch.Tensor, typing.Any]]prediction\_loss\_only: boolignore\_keys: typing.Optional[typing.List[str]] = None ) **→** Tuple[Optional[torch.Tensor], Optional[torch.Tensor], Optional[torch.Tensor]]

**Parameters**

* **model** (nn.Module) — The model to evaluate.
* **inputs** (Dict[str, Union[torch.Tensor, Any]]) — The inputs and targets of the model.

The dictionary will be unpacked before being fed to the model. Most models expect the targets under the argument labels. Check your model’s documentation for all accepted arguments.

* **prediction\_loss\_only** (bool) — Whether or not to return the loss only.
* **ignore\_keys** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.

**Returns**

**Tuple[Optional[torch.Tensor], Optional[torch.Tensor], Optional[torch.Tensor]]**

A tuple with the loss, logits and labels (each being optional).

Perform an evaluation step on model using inputs.

Subclass and override to inject custom behavior.

push\_to\_hub

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L3650)

( commit\_message: typing.Optional[str] = 'End of training'blocking: bool = True\*\*kwargs )

**Parameters**

* **commit\_message** (str, *optional*, defaults to "End of training") — Message to commit while pushing.
* **blocking** (bool, *optional*, defaults to True) — Whether the function should return only when the git push has finished.
* **kwargs** (Dict[str, Any], *optional*) — Additional keyword arguments passed along to [create\_model\_card()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.create_model_card).

Upload self.model and self.tokenizer to the 🤗 model hub on the repo self.args.hub\_model\_id.

remove\_callback

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L681)

( callback )

**Parameters**

* **callback** (type or [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback)) — A [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback) class or an instance of a [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback). In the first case, will remove the first member of that class found in the list of callbacks.

Remove a callback from the current list of [TrainerCallback](https://huggingface.co/docs/transformers/main/en/main_classes/callback#transformers.TrainerCallback).

save\_metrics

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_pt_utils.py#L995)

( splitmetricscombined = True )

**Parameters**

* **split** (str) — Mode/split name: one of train, eval, test, all
* **metrics** (Dict[str, float]) — The metrics returned from train/evaluate/predict
* **combined** (bool, *optional*, defaults to True) — Creates combined metrics by updating all\_results.json with metrics of this call

Save metrics into a json file for that split, e.g. train\_results.json.

Under distributed environment this is done only for a process with rank 0.

To understand the metrics please read the docstring of [log\_metrics()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.log_metrics). The only difference is that raw unformatted numbers are saved in the current method.

save\_model

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2806)

( output\_dir: typing.Optional[str] = None\_internal\_call: bool = False )

Will save the model, so you can reload it using from\_pretrained().

Will only save from the main process.

save\_state

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_pt_utils.py#L1033)

( )

Saves the Trainer state, since Trainer.save\_model saves only the tokenizer with the model

Under distributed environment this is done only for a process with rank 0.

train

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L1438)

( resume\_from\_checkpoint: typing.Union[str, bool, NoneType] = Nonetrial: typing.Union[ForwardRef('optuna.Trial'), typing.Dict[str, typing.Any]] = Noneignore\_keys\_for\_eval: typing.Optional[typing.List[str]] = None\*\*kwargs )

**Parameters**

* **resume\_from\_checkpoint** (str or bool, *optional*) — If a str, local path to a saved checkpoint as saved by a previous instance of [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer). If a bool and equals True, load the last checkpoint in *args.output\_dir* as saved by a previous instance of [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer). If present, training will resume from the model/optimizer/scheduler states loaded here.
* **trial** (optuna.Trial or Dict[str, Any], *optional*) — The trial run or the hyperparameter dictionary for hyperparameter search.
* **ignore\_keys\_for\_eval** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions for evaluation during the training.
* **kwargs** (Dict[str, Any], *optional*) — Additional keyword arguments used to hide deprecated arguments

Main training entry point.

training\_step

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer.py#L2711)

( model: Moduleinputs: typing.Dict[str, typing.Union[torch.Tensor, typing.Any]] ) **→** torch.Tensor

**Parameters**

* **model** (nn.Module) — The model to train.
* **inputs** (Dict[str, Union[torch.Tensor, Any]]) — The inputs and targets of the model.

The dictionary will be unpacked before being fed to the model. Most models expect the targets under the argument labels. Check your model’s documentation for all accepted arguments.

**Returns**

torch.Tensor

The tensor with training loss on this batch.

Perform a training step on a batch of inputs.

Subclass and override to inject custom behavior.

**Seq2SeqTrainer**

classtransformers.**Seq2SeqTrainer**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_seq2seq.py#L41)

( model: typing.Union[ForwardRef('PreTrainedModel'), torch.nn.modules.module.Module] = Noneargs: TrainingArguments = Nonedata\_collator: typing.Optional[ForwardRef('DataCollator')] = Nonetrain\_dataset: typing.Optional[torch.utils.data.dataset.Dataset] = Noneeval\_dataset: typing.Union[torch.utils.data.dataset.Dataset, typing.Dict[str, torch.utils.data.dataset.Dataset], NoneType] = Nonetokenizer: typing.Optional[ForwardRef('PreTrainedTokenizerBase')] = Nonemodel\_init: typing.Union[typing.Callable[[], ForwardRef('PreTrainedModel')], NoneType] = Nonecompute\_metrics: typing.Union[typing.Callable[[ForwardRef('EvalPrediction')], typing.Dict], NoneType] = Nonecallbacks: typing.Optional[typing.List[ForwardRef('TrainerCallback')]] = Noneoptimizers: typing.Tuple[torch.optim.optimizer.Optimizer, torch.optim.lr\_scheduler.LambdaLR] = (None, None)preprocess\_logits\_for\_metrics: typing.Union[typing.Callable[[torch.Tensor, torch.Tensor], torch.Tensor], NoneType] = None )

evaluate

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_seq2seq.py#L112)

( eval\_dataset: typing.Optional[torch.utils.data.dataset.Dataset] = Noneignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'eval'\*\*gen\_kwargs )

**Parameters**

* **eval\_dataset** (Dataset, *optional*) — Pass a dataset if you wish to override self.eval\_dataset. If it is an [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. It must implement the \_\_len\_\_ method.
* **ignore\_keys** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.
* **metric\_key\_prefix** (str, *optional*, defaults to "eval") — An optional prefix to be used as the metrics key prefix. For example the metrics “bleu” will be named “eval\_bleu” if the prefix is "eval" (default)
* **max\_length** (int, *optional*) — The maximum target length to use when predicting with the generate method.
* **num\_beams** (int, *optional*) — Number of beams for beam search that will be used when predicting with the generate method. 1 means no beam search. gen\_kwargs — Additional generate specific kwargs.

Run evaluation and returns metrics.

The calling script will be responsible for providing a method to compute metrics, as they are task-dependent (pass it to the init compute\_metrics argument).

You can also subclass and override this method to inject custom behavior.

predict

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/trainer_seq2seq.py#L168)

( test\_dataset: Datasetignore\_keys: typing.Optional[typing.List[str]] = Nonemetric\_key\_prefix: str = 'test'\*\*gen\_kwargs )

**Parameters**

* **test\_dataset** (Dataset) — Dataset to run the predictions on. If it is a [Dataset](https://huggingface.co/docs/datasets/main/en/package_reference/main_classes#datasets.Dataset), columns not accepted by the model.forward() method are automatically removed. Has to implement the method \_\_len\_\_
* **ignore\_keys** (List[str], *optional*) — A list of keys in the output of your model (if it is a dictionary) that should be ignored when gathering predictions.
* **metric\_key\_prefix** (str, *optional*, defaults to "eval") — An optional prefix to be used as the metrics key prefix. For example the metrics “bleu” will be named “eval\_bleu” if the prefix is "eval" (default)
* **max\_length** (int, *optional*) — The maximum target length to use when predicting with the generate method.
* **num\_beams** (int, *optional*) — Number of beams for beam search that will be used when predicting with the generate method. 1 means no beam search. gen\_kwargs — Additional generate specific kwargs.

Run prediction and returns predictions and potential metrics.

Depending on the dataset and your use case, your test dataset may contain labels. In that case, this method will also return metrics, like in evaluate().

If your predictions or labels have different sequence lengths (for instance because you’re doing dynamic padding in a token classification task) the predictions will be padded (on the right) to allow for concatenation into one array. The padding index is -100.

Returns: *NamedTuple* A namedtuple with the following keys:

* predictions (np.ndarray): The predictions on test\_dataset.
* label\_ids (np.ndarray, *optional*): The labels (if the dataset contained some).
* metrics (Dict[str, float], *optional*): The potential dictionary of metrics (if the dataset contained labels).

**TrainingArguments**

classtransformers.**TrainingArguments**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L161)

( output\_dir: stroverwrite\_output\_dir: bool = Falsedo\_train: bool = Falsedo\_eval: bool = Falsedo\_predict: bool = Falseevaluation\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'no'prediction\_loss\_only: bool = Falseper\_device\_train\_batch\_size: int = 8per\_device\_eval\_batch\_size: int = 8per\_gpu\_train\_batch\_size: typing.Optional[int] = Noneper\_gpu\_eval\_batch\_size: typing.Optional[int] = Nonegradient\_accumulation\_steps: int = 1eval\_accumulation\_steps: typing.Optional[int] = Noneeval\_delay: typing.Optional[float] = 0learning\_rate: float = 5e-05weight\_decay: float = 0.0adam\_beta1: float = 0.9adam\_beta2: float = 0.999adam\_epsilon: float = 1e-08max\_grad\_norm: float = 1.0num\_train\_epochs: float = 3.0max\_steps: int = -1lr\_scheduler\_type: typing.Union[transformers.trainer\_utils.SchedulerType, str] = 'linear'lr\_scheduler\_kwargs: typing.Optional[typing.Dict] = <factory>warmup\_ratio: float = 0.0warmup\_steps: int = 0log\_level: typing.Optional[str] = 'passive'log\_level\_replica: typing.Optional[str] = 'warning'log\_on\_each\_node: bool = Truelogging\_dir: typing.Optional[str] = Nonelogging\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'steps'logging\_first\_step: bool = Falselogging\_steps: float = 500logging\_nan\_inf\_filter: bool = Truesave\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'steps'save\_steps: float = 500save\_total\_limit: typing.Optional[int] = Nonesave\_safetensors: typing.Optional[bool] = Truesave\_on\_each\_node: bool = Falsesave\_only\_model: bool = Falseno\_cuda: bool = Falseuse\_cpu: bool = Falseuse\_mps\_device: bool = Falseseed: int = 42data\_seed: typing.Optional[int] = Nonejit\_mode\_eval: bool = Falseuse\_ipex: bool = Falsebf16: bool = Falsefp16: bool = Falsefp16\_opt\_level: str = 'O1'half\_precision\_backend: str = 'auto'bf16\_full\_eval: bool = Falsefp16\_full\_eval: bool = Falsetf32: typing.Optional[bool] = Nonelocal\_rank: int = -1ddp\_backend: typing.Optional[str] = Nonetpu\_num\_cores: typing.Optional[int] = Nonetpu\_metrics\_debug: bool = Falsedebug: typing.Union[str, typing.List[transformers.debug\_utils.DebugOption]] = ''dataloader\_drop\_last: bool = Falseeval\_steps: typing.Optional[float] = Nonedataloader\_num\_workers: int = 0past\_index: int = -1run\_name: typing.Optional[str] = Nonedisable\_tqdm: typing.Optional[bool] = Noneremove\_unused\_columns: typing.Optional[bool] = Truelabel\_names: typing.Optional[typing.List[str]] = Noneload\_best\_model\_at\_end: typing.Optional[bool] = Falsemetric\_for\_best\_model: typing.Optional[str] = Nonegreater\_is\_better: typing.Optional[bool] = Noneignore\_data\_skip: bool = Falsefsdp: typing.Union[typing.List[transformers.trainer\_utils.FSDPOption], str, NoneType] = ''fsdp\_min\_num\_params: int = 0fsdp\_config: typing.Optional[str] = Nonefsdp\_transformer\_layer\_cls\_to\_wrap: typing.Optional[str] = Nonedeepspeed: typing.Optional[str] = Nonelabel\_smoothing\_factor: float = 0.0optim: typing.Union[transformers.training\_args.OptimizerNames, str] = 'adamw\_torch'optim\_args: typing.Optional[str] = Noneadafactor: bool = Falsegroup\_by\_length: bool = Falselength\_column\_name: typing.Optional[str] = 'length'report\_to: typing.Optional[typing.List[str]] = Noneddp\_find\_unused\_parameters: typing.Optional[bool] = Noneddp\_bucket\_cap\_mb: typing.Optional[int] = Noneddp\_broadcast\_buffers: typing.Optional[bool] = Nonedataloader\_pin\_memory: bool = Truedataloader\_persistent\_workers: bool = Falseskip\_memory\_metrics: bool = Trueuse\_legacy\_prediction\_loop: bool = Falsepush\_to\_hub: bool = Falseresume\_from\_checkpoint: typing.Optional[str] = Nonehub\_model\_id: typing.Optional[str] = Nonehub\_strategy: typing.Union[transformers.trainer\_utils.HubStrategy, str] = 'every\_save'hub\_token: typing.Optional[str] = Nonehub\_private\_repo: bool = Falsehub\_always\_push: bool = Falsegradient\_checkpointing: bool = Falsegradient\_checkpointing\_kwargs: typing.Optional[dict] = Noneinclude\_inputs\_for\_metrics: bool = Falsefp16\_backend: str = 'auto'push\_to\_hub\_model\_id: typing.Optional[str] = Nonepush\_to\_hub\_organization: typing.Optional[str] = Nonepush\_to\_hub\_token: typing.Optional[str] = Nonemp\_parameters: str = ''auto\_find\_batch\_size: bool = Falsefull\_determinism: bool = Falsetorchdynamo: typing.Optional[str] = Noneray\_scope: typing.Optional[str] = 'last'ddp\_timeout: typing.Optional[int] = 1800torch\_compile: bool = Falsetorch\_compile\_backend: typing.Optional[str] = Nonetorch\_compile\_mode: typing.Optional[str] = Nonedispatch\_batches: typing.Optional[bool] = Nonesplit\_batches: typing.Optional[bool] = Falseinclude\_tokens\_per\_second: typing.Optional[bool] = Falseinclude\_num\_input\_tokens\_seen: typing.Optional[bool] = Falseneftune\_noise\_alpha: float = None )

Expand 105 parameters

**Parameters**

* **output\_dir** (str) — The output directory where the model predictions and checkpoints will be written.
* **overwrite\_output\_dir** (bool, *optional*, defaults to False) — If True, overwrite the content of the output directory. Use this to continue training if output\_dir points to a checkpoint directory.
* **do\_train** (bool, *optional*, defaults to False) — Whether to run training or not. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **do\_eval** (bool, *optional*) — Whether to run evaluation on the validation set or not. Will be set to True if evaluation\_strategy is different from "no". This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **do\_predict** (bool, *optional*, defaults to False) — Whether to run predictions on the test set or not. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **evaluation\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "no") — The evaluation strategy to adopt during training. Possible values are:
  + "no": No evaluation is done during training.
  + "steps": Evaluation is done (and logged) every eval\_steps.
  + "epoch": Evaluation is done at the end of each epoch.
* **prediction\_loss\_only** (bool, *optional*, defaults to False) — When performing evaluation and generating predictions, only returns the loss.
* **per\_device\_train\_batch\_size** (int, *optional*, defaults to 8) — The batch size per GPU/XPU/TPU/MPS/NPU core/CPU for training.
* **per\_device\_eval\_batch\_size** (int, *optional*, defaults to 8) — The batch size per GPU/XPU/TPU/MPS/NPU core/CPU for evaluation.
* **gradient\_accumulation\_steps** (int, *optional*, defaults to 1) — Number of updates steps to accumulate the gradients for, before performing a backward/update pass.

When using gradient accumulation, one step is counted as one step with backward pass. Therefore, logging, evaluation, save will be conducted every gradient\_accumulation\_steps \* xxx\_step training examples.

* **eval\_accumulation\_steps** (int, *optional*) — Number of predictions steps to accumulate the output tensors for, before moving the results to the CPU. If left unset, the whole predictions are accumulated on GPU/NPU/TPU before being moved to the CPU (faster but requires more memory).
* **eval\_delay** (float, *optional*) — Number of epochs or steps to wait for before the first evaluation can be performed, depending on the evaluation\_strategy.
* **learning\_rate** (float, *optional*, defaults to 5e-5) — The initial learning rate for [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **weight\_decay** (float, *optional*, defaults to 0) — The weight decay to apply (if not zero) to all layers except all bias and LayerNorm weights in [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_beta1** (float, *optional*, defaults to 0.9) — The beta1 hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_beta2** (float, *optional*, defaults to 0.999) — The beta2 hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_epsilon** (float, *optional*, defaults to 1e-8) — The epsilon hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **max\_grad\_norm** (float, *optional*, defaults to 1.0) — Maximum gradient norm (for gradient clipping).
* **num\_train\_epochs(**float**,** *optional*, defaults to 3.0) — Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).
* **max\_steps** (int, *optional*, defaults to -1) — If set to a positive number, the total number of training steps to perform. Overrides num\_train\_epochs. For a finite dataset, training is reiterated through the dataset (if all data is exhausted) until max\_steps is reached.
* **lr\_scheduler\_type** (str or [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType), *optional*, defaults to "linear") — The scheduler type to use. See the documentation of [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType) for all possible values.
* **lr\_scheduler\_kwargs** (‘dict’, *optional*, defaults to {}) — The extra arguments for the lr\_scheduler. See the documentation of each scheduler for possible values.
* **warmup\_ratio** (float, *optional*, defaults to 0.0) — Ratio of total training steps used for a linear warmup from 0 to learning\_rate.
* **warmup\_steps** (int, *optional*, defaults to 0) — Number of steps used for a linear warmup from 0 to learning\_rate. Overrides any effect of warmup\_ratio.
* **log\_level** (str, *optional*, defaults to passive) — Logger log level to use on the main process. Possible choices are the log levels as strings: ‘debug’, ‘info’, ‘warning’, ‘error’ and ‘critical’, plus a ‘passive’ level which doesn’t set anything and keeps the current log level for the Transformers library (which will be "warning" by default).
* **log\_level\_replica** (str, *optional*, defaults to "warning") — Logger log level to use on replicas. Same choices as log\_level”
* **log\_on\_each\_node** (bool, *optional*, defaults to True) — In multinode distributed training, whether to log using log\_level once per node, or only on the main node.
* **logging\_dir** (str, *optional*) — [TensorBoard](https://www.tensorflow.org/tensorboard) log directory. Will default to \*output\_dir/runs/**CURRENT\_DATETIME\_HOSTNAME\***.
* **logging\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The logging strategy to adopt during training. Possible values are:
  + "no": No logging is done during training.
  + "epoch": Logging is done at the end of each epoch.
  + "steps": Logging is done every logging\_steps.
* **logging\_first\_step** (bool, *optional*, defaults to False) — Whether to log and evaluate the first global\_step or not.
* **logging\_steps** (int or float, *optional*, defaults to 500) — Number of update steps between two logs if logging\_strategy="steps". Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **logging\_nan\_inf\_filter** (bool, *optional*, defaults to True) — Whether to filter nan and inf losses for logging. If set to True the loss of every step that is nan or inf is filtered and the average loss of the current logging window is taken instead.

logging\_nan\_inf\_filter only influences the logging of loss values, it does not change the behavior the gradient is computed or applied to the model.

* **save\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The checkpoint save strategy to adopt during training. Possible values are:
  + "no": No save is done during training.
  + "epoch": Save is done at the end of each epoch.
  + "steps": Save is done every save\_steps.
* **save\_steps** (int or float, *optional*, defaults to 500) — Number of updates steps before two checkpoint saves if save\_strategy="steps". Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **save\_total\_limit** (int, *optional*) — If a value is passed, will limit the total amount of checkpoints. Deletes the older checkpoints in output\_dir. When load\_best\_model\_at\_end is enabled, the “best” checkpoint according to metric\_for\_best\_model will always be retained in addition to the most recent ones. For example, for save\_total\_limit=5 and load\_best\_model\_at\_end, the four last checkpoints will always be retained alongside the best model. When save\_total\_limit=1 and load\_best\_model\_at\_end, it is possible that two checkpoints are saved: the last one and the best one (if they are different).
* **save\_safetensors** (bool, *optional*, defaults to True) — Use [safetensors](https://huggingface.co/docs/safetensors) saving and loading for state dicts instead of default torch.load and torch.save.
* **save\_on\_each\_node** (bool, *optional*, defaults to False) — When doing multi-node distributed training, whether to save models and checkpoints on each node, or only on the main one.

This should not be activated when the different nodes use the same storage as the files will be saved with the same names for each node.

* **save\_only\_model** (bool, *optional*, defaults to False) — When checkpointing, whether to only save the model, or also the optimizer, scheduler & rng state. Note that when this is true, you won’t be able to resume training from checkpoint. This enables you to save storage by not storing the optimizer, scheduler & rng state. You can only load the model using from\_pretrained with this option set to True.
* **use\_cpu** (bool, *optional*, defaults to False) — Whether or not to use cpu. If set to False, we will use cuda or mps device if available.
* **seed** (int, *optional*, defaults to 42) — Random seed that will be set at the beginning of training. To ensure reproducibility across runs, use the ~Trainer.model\_init function to instantiate the model if it has some randomly initialized parameters.
* **data\_seed** (int, *optional*) — Random seed to be used with data samplers. If not set, random generators for data sampling will use the same seed as seed. This can be used to ensure reproducibility of data sampling, independent of the model seed.
* **jit\_mode\_eval** (bool, *optional*, defaults to False) — Whether or not to use PyTorch jit trace for inference.
* **use\_ipex** (bool, *optional*, defaults to False) — Use Intel extension for PyTorch when it is available. [IPEX installation](https://github.com/intel/intel-extension-for-pytorch).
* **bf16** (bool, *optional*, defaults to False) — Whether to use bf16 16-bit (mixed) precision training instead of 32-bit training. Requires Ampere or higher NVIDIA architecture or using CPU (use\_cpu) or Ascend NPU. This is an experimental API and it may change.
* **fp16** (bool, *optional*, defaults to False) — Whether to use fp16 16-bit (mixed) precision training instead of 32-bit training.
* **fp16\_opt\_level** (str, *optional*, defaults to ‘O1’) — For fp16 training, Apex AMP optimization level selected in [‘O0’, ‘O1’, ‘O2’, and ‘O3’]. See details on the [Apex documentation](https://nvidia.github.io/apex/amp).
* **fp16\_backend** (str, *optional*, defaults to "auto") — This argument is deprecated. Use half\_precision\_backend instead.
* **half\_precision\_backend** (str, *optional*, defaults to "auto") — The backend to use for mixed precision training. Must be one of "auto", "apex", "cpu\_amp". "auto" will use CPU/CUDA AMP or APEX depending on the PyTorch version detected, while the other choices will force the requested backend.
* **bf16\_full\_eval** (bool, *optional*, defaults to False) — Whether to use full bfloat16 evaluation instead of 32-bit. This will be faster and save memory but can harm metric values. This is an experimental API and it may change.
* **fp16\_full\_eval** (bool, *optional*, defaults to False) — Whether to use full float16 evaluation instead of 32-bit. This will be faster and save memory but can harm metric values.
* **tf32** (bool, *optional*) — Whether to enable the TF32 mode, available in Ampere and newer GPU architectures. The default value depends on PyTorch’s version default of torch.backends.cuda.matmul.allow\_tf32. For more details please refer to the [TF32](https://huggingface.co/docs/transformers/performance#tf32) documentation. This is an experimental API and it may change.
* **local\_rank** (int, *optional*, defaults to -1) — Rank of the process during distributed training.
* **ddp\_backend** (str, *optional*) — The backend to use for distributed training. Must be one of "nccl", "mpi", "ccl", "gloo", "hccl".
* **tpu\_num\_cores** (int, *optional*) — When training on TPU, the number of TPU cores (automatically passed by launcher script).
* **dataloader\_drop\_last** (bool, *optional*, defaults to False) — Whether to drop the last incomplete batch (if the length of the dataset is not divisible by the batch size) or not.
* **eval\_steps** (int or float, *optional*) — Number of update steps between two evaluations if evaluation\_strategy="steps". Will default to the same value as logging\_steps if not set. Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **dataloader\_num\_workers** (int, *optional*, defaults to 0) — Number of subprocesses to use for data loading (PyTorch only). 0 means that the data will be loaded in the main process.
* **past\_index** (int, *optional*, defaults to -1) — Some models like [TransformerXL](https://huggingface.co/docs/transformers/main/model_doc/transformerxl) or [XLNet](https://huggingface.co/docs/transformers/main/model_doc/xlnet) can make use of the past hidden states for their predictions. If this argument is set to a positive int, the Trainer will use the corresponding output (usually index 2) as the past state and feed it to the model at the next training step under the keyword argument mems.
* **run\_name** (str, *optional*) — A descriptor for the run. Typically used for [wandb](https://www.wandb.com/) and [mlflow](https://www.mlflow.org/) logging.
* **disable\_tqdm** (bool, *optional*) — Whether or not to disable the tqdm progress bars and table of metrics produced by ~notebook.NotebookTrainingTracker in Jupyter Notebooks. Will default to True if the logging level is set to warn or lower (default), False otherwise.
* **remove\_unused\_columns** (bool, *optional*, defaults to True) — Whether or not to automatically remove the columns unused by the model forward method.

(Note that this behavior is not implemented for TFTrainer yet.)

* **label\_names** (List[str], *optional*) — The list of keys in your dictionary of inputs that correspond to the labels.

Will eventually default to the list of argument names accepted by the model that contain the word “label”, except if the model used is one of the XxxForQuestionAnswering in which case it will also include the ["start\_positions", "end\_positions"] keys.

* **load\_best\_model\_at\_end** (bool, *optional*, defaults to False) — Whether or not to load the best model found during training at the end of training. When this option is enabled, the best checkpoint will always be saved. See [save\_total\_limit](https://huggingface.co/docs/transformers/main_classes/trainer#transformers.TrainingArguments.save_total_limit) for more.

When set to True, the parameters save\_strategy needs to be the same as evaluation\_strategy, and in the case it is “steps”, save\_steps must be a round multiple of eval\_steps.

* **metric\_for\_best\_model** (str, *optional*) — Use in conjunction with load\_best\_model\_at\_end to specify the metric to use to compare two different models. Must be the name of a metric returned by the evaluation with or without the prefix "eval\_". Will default to "loss" if unspecified and load\_best\_model\_at\_end=True (to use the evaluation loss).

If you set this value, greater\_is\_better will default to True. Don’t forget to set it to False if your metric is better when lower.

* **greater\_is\_better** (bool, *optional*) — Use in conjunction with load\_best\_model\_at\_end and metric\_for\_best\_model to specify if better models should have a greater metric or not. Will default to:
  + True if metric\_for\_best\_model is set to a value that isn’t "loss" or "eval\_loss".
  + False if metric\_for\_best\_model is not set, or set to "loss" or "eval\_loss".
* **ignore\_data\_skip** (bool, *optional*, defaults to False) — When resuming training, whether or not to skip the epochs and batches to get the data loading at the same stage as in the previous training. If set to True, the training will begin faster (as that skipping step can take a long time) but will not yield the same results as the interrupted training would have.
* **fsdp** (bool, str or list of FSDPOption, *optional*, defaults to '') — Use PyTorch Distributed Parallel Training (in distributed training only).

A list of options along the following:

* + "full\_shard": Shard parameters, gradients and optimizer states.
  + "shard\_grad\_op": Shard optimizer states and gradients.
  + "hybrid\_shard": Apply FULL\_SHARD within a node, and replicate parameters across nodes.
  + "hybrid\_shard\_zero2": Apply SHARD\_GRAD\_OP within a node, and replicate parameters across nodes.
  + "offload": Offload parameters and gradients to CPUs (only compatible with "full\_shard" and "shard\_grad\_op").
  + "auto\_wrap": Automatically recursively wrap layers with FSDP using default\_auto\_wrap\_policy.
* **fsdp\_config** (str or dict, *optional*) — Config to be used with fsdp (Pytorch Distributed Parallel Training). The value is either a location of fsdp json config file (e.g., fsdp\_config.json) or an already loaded json file as dict.

A List of config and its options:

* + min\_num\_params (int, *optional*, defaults to 0): FSDP’s minimum number of parameters for Default Auto Wrapping. (useful only when fsdp field is passed).
  + transformer\_layer\_cls\_to\_wrap (List[str], *optional*): List of transformer layer class names (case-sensitive) to wrap, e.g, BertLayer, GPTJBlock, T5Block … (useful only when fsdp flag is passed).
  + backward\_prefetch (str, *optional*) FSDP’s backward prefetch mode. Controls when to prefetch next set of parameters (useful only when fsdp field is passed).

A list of options along the following:

* + - "backward\_pre" : Prefetches the next set of parameters before the current set of parameter’s gradient computation.
    - "backward\_post" : This prefetches the next set of parameters after the current set of parameter’s gradient computation.
  + forward\_prefetch (bool, *optional*, defaults to False) FSDP’s forward prefetch mode (useful only when fsdp field is passed). If "True", then FSDP explicitly prefetches the next upcoming all-gather while executing in the forward pass.
  + limit\_all\_gathers (bool, *optional*, defaults to False) FSDP’s limit\_all\_gathers (useful only when fsdp field is passed). If "True", FSDP explicitly synchronizes the CPU thread to prevent too many in-flight all-gathers.
  + use\_orig\_params (bool, *optional*, defaults to True) If "True", allows non-uniform requires\_grad during init, which means support for interspersed frozen and trainable paramteres. Useful in cases such as parameter-efficient fine-tuning. Please refer this [blog](<https://dev-discuss.pytorch.org/t/rethinking-pytorch-fully-sharded-data-parallel-fsdp-from-first-principles/1019>
  + sync\_module\_states (bool, *optional*, defaults to True) If "True", each individually wrapped FSDP unit will broadcast module parameters from rank 0 to ensure they are the same across all ranks after initialization
  + activation\_checkpointing (bool, *optional*, defaults to False): If "True", activation checkpointing is a technique to reduce memory usage by clearing activations of certain layers and recomputing them during a backward pass. Effectively, this trades extra computation time for reduced memory usage.
  + xla (bool, *optional*, defaults to False): Whether to use PyTorch/XLA Fully Sharded Data Parallel Training. This is an experimental feature and its API may evolve in the future.
  + xla\_fsdp\_settings (dict, *optional*) The value is a dictionary which stores the XLA FSDP wrapping parameters.

For a complete list of options, please see [here](https://github.com/pytorch/xla/blob/master/torch_xla/distributed/fsdp/xla_fully_sharded_data_parallel.py).

* + xla\_fsdp\_grad\_ckpt (bool, *optional*, defaults to False): Will use gradient checkpointing over each nested XLA FSDP wrapped layer. This setting can only be used when the xla flag is set to true, and an auto wrapping policy is specified through fsdp\_min\_num\_params or fsdp\_transformer\_layer\_cls\_to\_wrap.
* **deepspeed** (str or dict, *optional*) — Use [Deepspeed](https://github.com/microsoft/deepspeed). This is an experimental feature and its API may evolve in the future. The value is either the location of DeepSpeed json config file (e.g., ds\_config.json) or an already loaded json file as a dict”
* **label\_smoothing\_factor** (float, *optional*, defaults to 0.0) — The label smoothing factor to use. Zero means no label smoothing, otherwise the underlying onehot-encoded labels are changed from 0s and 1s to label\_smoothing\_factor/num\_labels and 1 - label\_smoothing\_factor + label\_smoothing\_factor/num\_labels respectively.
* **debug** (str or list of DebugOption, *optional*, defaults to "") — Enable one or more debug features. This is an experimental feature.

Possible options are:

* + "underflow\_overflow": detects overflow in model’s input/outputs and reports the last frames that led to the event
  + "tpu\_metrics\_debug": print debug metrics on TPU

The options should be separated by whitespaces.

* **optim** (str or training\_args.OptimizerNames, *optional*, defaults to "adamw\_torch") — The optimizer to use: adamw\_hf, adamw\_torch, adamw\_torch\_fused, adamw\_apex\_fused, adamw\_anyprecision or adafactor.
* **optim\_args** (str, *optional*) — Optional arguments that are supplied to AnyPrecisionAdamW.
* **group\_by\_length** (bool, *optional*, defaults to False) — Whether or not to group together samples of roughly the same length in the training dataset (to minimize padding applied and be more efficient). Only useful if applying dynamic padding.
* **length\_column\_name** (str, *optional*, defaults to "length") — Column name for precomputed lengths. If the column exists, grouping by length will use these values rather than computing them on train startup. Ignored unless group\_by\_length is True and the dataset is an instance of Dataset.
* **report\_to** (str or List[str], *optional*, defaults to "all") — The list of integrations to report the results and logs to. Supported platforms are "azure\_ml", "clearml", "codecarbon", "comet\_ml", "dagshub", "dvclive", "flyte", "mlflow", "neptune", "tensorboard", and "wandb". Use "all" to report to all integrations installed, "none" for no integrations.
* **ddp\_find\_unused\_parameters** (bool, *optional*) — When using distributed training, the value of the flag find\_unused\_parameters passed to DistributedDataParallel. Will default to False if gradient checkpointing is used, True otherwise.
* **ddp\_bucket\_cap\_mb** (int, *optional*) — When using distributed training, the value of the flag bucket\_cap\_mb passed to DistributedDataParallel.
* **ddp\_broadcast\_buffers** (bool, *optional*) — When using distributed training, the value of the flag broadcast\_buffers passed to DistributedDataParallel. Will default to False if gradient checkpointing is used, True otherwise.
* **dataloader\_pin\_memory** (bool, *optional*, defaults to True) — Whether you want to pin memory in data loaders or not. Will default to True.
* **dataloader\_persistent\_workers** (bool, *optional*, defaults to False) — If True, the data loader will not shut down the worker processes after a dataset has been consumed once. This allows to maintain the workers Dataset instances alive. Can potentially speed up training, but will increase RAM usage. Will default to False.
* **skip\_memory\_metrics** (bool, *optional*, defaults to True) — Whether to skip adding of memory profiler reports to metrics. This is skipped by default because it slows down the training and evaluation speed.
* **push\_to\_hub** (bool, *optional*, defaults to False) — Whether or not to push the model to the Hub every time the model is saved. If this is activated, output\_dir will begin a git directory synced with the repo (determined by hub\_model\_id) and the content will be pushed each time a save is triggered (depending on your save\_strategy). Calling [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) will also trigger a push.

If output\_dir exists, it needs to be a local clone of the repository to which the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) will be pushed.

* **resume\_from\_checkpoint** (str, *optional*) — The path to a folder with a valid checkpoint for your model. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **hub\_model\_id** (str, *optional*) — The name of the repository to keep in sync with the local *output\_dir*. It can be a simple model ID in which case the model will be pushed in your namespace. Otherwise it should be the whole repository name, for instance "user\_name/model", which allows you to push to an organization you are a member of with "organization\_name/model". Will default to user\_name/output\_dir\_name with *output\_dir\_name* being the name of output\_dir.

Will default to the name of output\_dir.

* **hub\_strategy** (str or HubStrategy, *optional*, defaults to "every\_save") — Defines the scope of what is pushed to the Hub and when. Possible values are:
  + "end": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card when the [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) method is called.
  + "every\_save": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card each time there is a model save. The pushes are asynchronous to not block training, and in case the save are very frequent, a new push is only attempted if the previous one is finished. A last push is made with the final model at the end of training.
  + "checkpoint": like "every\_save" but the latest checkpoint is also pushed in a subfolder named last-checkpoint, allowing you to resume training easily with trainer.train(resume\_from\_checkpoint="last-checkpoint").
  + "all\_checkpoints": like "checkpoint" but all checkpoints are pushed like they appear in the output folder (so you will get one checkpoint folder per folder in your final repository)
* **hub\_token** (str, *optional*) — The token to use to push the model to the Hub. Will default to the token in the cache folder obtained with huggingface-cli login.
* **hub\_private\_repo** (bool, *optional*, defaults to False) — If True, the Hub repo will be set to private.
* **hub\_always\_push** (bool, *optional*, defaults to False) — Unless this is True, the Trainer will skip pushing a checkpoint when the previous push is not finished.
* **gradient\_checkpointing** (bool, *optional*, defaults to False) — If True, use gradient checkpointing to save memory at the expense of slower backward pass.
* **gradient\_checkpointing\_kwargs** (dict, *optional*, defaults to None) — Key word arguments to be passed to the gradient\_checkpointing\_enable method.
* **include\_inputs\_for\_metrics** (bool, *optional*, defaults to False) — Whether or not the inputs will be passed to the compute\_metrics function. This is intended for metrics that need inputs, predictions and references for scoring calculation in Metric class.
* **auto\_find\_batch\_size** (bool, *optional*, defaults to False) — Whether to find a batch size that will fit into memory automatically through exponential decay, avoiding CUDA Out-of-Memory errors. Requires accelerate to be installed (pip install accelerate)
* **full\_determinism** (bool, *optional*, defaults to False) — If True, [enable\_full\_determinism()](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.enable_full_determinism) is called instead of [set\_seed()](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.set_seed) to ensure reproducible results in distributed training. Important: this will negatively impact the performance, so only use it for debugging.
* **torchdynamo** (str, *optional*) — If set, the backend compiler for TorchDynamo. Possible choices are "eager", "aot\_eager", "inductor", "nvfuser", "aot\_nvfuser", "aot\_cudagraphs", "ofi", "fx2trt", "onnxrt" and "ipex".
* **ray\_scope** (str, *optional*, defaults to "last") — The scope to use when doing hyperparameter search with Ray. By default, "last" will be used. Ray will then use the last checkpoint of all trials, compare those, and select the best one. However, other options are also available. See the [Ray documentation](https://docs.ray.io/en/latest/tune/api_docs/analysis.html#ray.tune.ExperimentAnalysis.get_best_trial) for more options.
* **ddp\_timeout** (int, *optional*, defaults to 1800) — The timeout for torch.distributed.init\_process\_group calls, used to avoid GPU socket timeouts when performing slow operations in distributed runnings. Please refer the [PyTorch documentation] (<https://pytorch.org/docs/stable/distributed.html#torch.distributed.init_process_group>) for more information.
* **use\_mps\_device** (bool, *optional*, defaults to False) — This argument is deprecated.mps device will be used if it is available similar to cuda device.
* **torch\_compile** (bool, *optional*, defaults to False) — Whether or not to compile the model using PyTorch 2.0 [torch.compile](https://pytorch.org/get-started/pytorch-2.0/).

This will use the best defaults for the [torch.compile API](https://pytorch.org/docs/stable/generated/torch.compile.html?highlight=torch+compile#torch.compile). You can customize the defaults with the argument torch\_compile\_backend and torch\_compile\_mode but we don’t guarantee any of them will work as the support is progressively rolled in in PyTorch.

This flag and the whole compile API is experimental and subject to change in future releases.

* **torch\_compile\_backend** (str, *optional*) — The backend to use in torch.compile. If set to any value, torch\_compile will be set to True.

Refer to the PyTorch doc for possible values and note that they may change across PyTorch versions.

This flag is experimental and subject to change in future releases.

* **torch\_compile\_mode** (str, *optional*) — The mode to use in torch.compile. If set to any value, torch\_compile will be set to True.

Refer to the PyTorch doc for possible values and note that they may change across PyTorch versions.

This flag is experimental and subject to change in future releases.

* **split\_batches** (bool, *optional*) — Whether or not the accelerator should split the batches yielded by the dataloaders across the devices during distributed training. If

set to True, the actual batch size used will be the same on any kind of distributed processes, but it must be a

round multiple of the number of processes you are using (such as GPUs).

* **include\_tokens\_per\_second** (bool, *optional*) — Whether or not to compute the number of tokens per second per device for training speed metrics.

This will iterate over the entire training dataloader once beforehand,

and will slow down the entire process.

* **include\_num\_input\_tokens\_seen** (bool, *optional*) — Whether or not to track the number of input tokens seen throughout training.

May be slower in distributed training as gather operations must be called.

* **neftune\_noise\_alpha** (Optional[float]) — If not None, this will activate NEFTune noise embeddings. This can drastically improve model performance for instruction fine-tuning. Check out the [original paper](https://arxiv.org/abs/2310.05914) and the [original code](https://github.com/neelsjain/NEFTune). Support transformers PreTrainedModel and also PeftModel from peft.

TrainingArguments is the subset of the arguments we use in our example scripts **which relate to the training loop itself**.

Using [HfArgumentParser](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.HfArgumentParser) we can turn this class into [argparse](https://docs.python.org/3/library/argparse#module-argparse) arguments that can be specified on the command line.

get\_process\_log\_level

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2048)

( )

Returns the log level to be used depending on whether this process is the main process of node 0, main process of node non-0, or a non-main process.

For the main process the log level defaults to the logging level set (logging.WARNING if you didn’t do anything) unless overridden by log\_level argument.

For the replica processes the log level defaults to logging.WARNING unless overridden by log\_level\_replica argument.

The choice between the main and replica process settings is made according to the return value of should\_log.

get\_warmup\_steps

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2137)

( num\_training\_steps: int )

Get number of steps used for a linear warmup.

main\_process\_first

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2086)

( local = Truedesc = 'work' )

**Parameters**

* **local** (bool, *optional*, defaults to True) — if True first means process of rank 0 of each node if False first means process of rank 0 of node rank 0 In multi-node environment with a shared filesystem you most likely will want to use local=False so that only the main process of the first node will do the processing. If however, the filesystem is not shared, then the main process of each node will need to do the processing, which is the default behavior.
* **desc** (str, *optional*, defaults to "work") — a work description to be used in debug logs

A context manager for torch distributed environment where on needs to do something on the main process, while blocking replicas, and when it’s finished releasing the replicas.

One such use is for datasets’s map feature which to be efficient should be run once on the main process, which upon completion saves a cached version of results and which then automatically gets loaded by the replicas.

set\_dataloader

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2649)

( train\_batch\_size: int = 8eval\_batch\_size: int = 8drop\_last: bool = Falsenum\_workers: int = 0pin\_memory: bool = Truepersistent\_workers: bool = Falseauto\_find\_batch\_size: bool = Falseignore\_data\_skip: bool = Falsesampler\_seed: typing.Optional[int] = None )

Expand 7 parameters

**Parameters**

* **drop\_last** (bool, *optional*, defaults to False) — Whether to drop the last incomplete batch (if the length of the dataset is not divisible by the batch size) or not.
* **num\_workers** (int, *optional*, defaults to 0) — Number of subprocesses to use for data loading (PyTorch only). 0 means that the data will be loaded in the main process.
* **pin\_memory** (bool, *optional*, defaults to True) — Whether you want to pin memory in data loaders or not. Will default to True.
* **persistent\_workers** (bool, *optional*, defaults to False) — If True, the data loader will not shut down the worker processes after a dataset has been consumed once. This allows to maintain the workers Dataset instances alive. Can potentially speed up training, but will increase RAM usage. Will default to False.
* **auto\_find\_batch\_size** (bool, *optional*, defaults to False) — Whether to find a batch size that will fit into memory automatically through exponential decay, avoiding CUDA Out-of-Memory errors. Requires accelerate to be installed (pip install accelerate)
* **ignore\_data\_skip** (bool, *optional*, defaults to False) — When resuming training, whether or not to skip the epochs and batches to get the data loading at the same stage as in the previous training. If set to True, the training will begin faster (as that skipping step can take a long time) but will not yield the same results as the interrupted training would have.
* **sampler\_seed** (int, *optional*) — Random seed to be used with data samplers. If not set, random generators for data sampling will use the same seed as self.seed. This can be used to ensure reproducibility of data sampling, independent of the model seed.

A method that regroups all arguments linked to the dataloaders creation.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_dataloader(train\_batch\_size=16, eval\_batch\_size=64)

>>> args.per\_device\_train\_batch\_size

16

set\_evaluate

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2258)

( strategy: typing.Union[str, transformers.trainer\_utils.IntervalStrategy] = 'no'steps: int = 500batch\_size: int = 8accumulation\_steps: typing.Optional[int] = Nonedelay: typing.Optional[float] = Noneloss\_only: bool = Falsejit\_mode: bool = False )

Expand 7 parameters

**Parameters**

* **strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "no") — The evaluation strategy to adopt during training. Possible values are:
  + "no": No evaluation is done during training.
  + "steps": Evaluation is done (and logged) every steps.
  + "epoch": Evaluation is done at the end of each epoch.

Setting a strategy different from "no" will set self.do\_eval to True.

* **steps** (int, *optional*, defaults to 500) — Number of update steps between two evaluations if strategy="steps".
* **batch\_size** (int *optional*, defaults to 8) — The batch size per device (GPU/TPU core/CPU…) used for evaluation.
* **accumulation\_steps** (int, *optional*) — Number of predictions steps to accumulate the output tensors for, before moving the results to the CPU. If left unset, the whole predictions are accumulated on GPU/TPU before being moved to the CPU (faster but requires more memory).
* **delay** (float, *optional*) — Number of epochs or steps to wait for before the first evaluation can be performed, depending on the evaluation\_strategy.
* **loss\_only** (bool, *optional*, defaults to False) — Ignores all outputs except the loss.
* **jit\_mode** (bool, *optional*) — Whether or not to use PyTorch jit trace for inference.

A method that regroups all arguments linked to evaluation.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_evaluate(strategy="steps", steps=100)

>>> args.eval\_steps

100

set\_logging

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2408)

( strategy: typing.Union[str, transformers.trainer\_utils.IntervalStrategy] = 'steps'steps: int = 500report\_to: typing.Union[str, typing.List[str]] = 'none'level: str = 'passive'first\_step: bool = Falsenan\_inf\_filter: bool = Falseon\_each\_node: bool = Falsereplica\_level: str = 'passive' )

Expand 8 parameters

**Parameters**

* **strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The logging strategy to adopt during training. Possible values are:
  + "no": No save is done during training.
  + "epoch": Save is done at the end of each epoch.
  + "steps": Save is done every save\_steps.
* **steps** (int, *optional*, defaults to 500) — Number of update steps between two logs if strategy="steps".
* **level** (str, *optional*, defaults to "passive") — Logger log level to use on the main process. Possible choices are the log levels as strings: "debug", "info", "warning", "error" and "critical", plus a "passive" level which doesn’t set anything and lets the application set the level.
* **report\_to** (str or List[str], *optional*, defaults to "all") — The list of integrations to report the results and logs to. Supported platforms are "azure\_ml", "clearml", "codecarbon", "comet\_ml", "dagshub", "dvclive", "flyte", "mlflow", "neptune", "tensorboard", and "wandb". Use "all" to report to all integrations installed, "none" for no integrations.
* **first\_step** (bool, *optional*, defaults to False) — Whether to log and evaluate the first global\_step or not.
* **nan\_inf\_filter** (bool, *optional*, defaults to True) — Whether to filter nan and inf losses for logging. If set to True the loss of every step that is nan or inf is filtered and the average loss of the current logging window is taken instead.

nan\_inf\_filter only influences the logging of loss values, it does not change the behavior the gradient is computed or applied to the model.

* **on\_each\_node** (bool, *optional*, defaults to True) — In multinode distributed training, whether to log using log\_level once per node, or only on the main node.
* **replica\_level** (str, *optional*, defaults to "passive") — Logger log level to use on replicas. Same choices as log\_level

A method that regroups all arguments linked to logging.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_logging(strategy="steps", steps=100)

>>> args.logging\_steps

100

set\_lr\_scheduler

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2604)

( name: typing.Union[str, transformers.trainer\_utils.SchedulerType] = 'linear'num\_epochs: float = 3.0max\_steps: int = -1warmup\_ratio: float = 0warmup\_steps: int = 0 )

**Parameters**

* **name** (str or [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType), *optional*, defaults to "linear") — The scheduler type to use. See the documentation of [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType) for all possible values.
* **num\_epochs(**float**,** *optional*, defaults to 3.0) — Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).
* **max\_steps** (int, *optional*, defaults to -1) — If set to a positive number, the total number of training steps to perform. Overrides num\_train\_epochs. For a finite dataset, training is reiterated through the dataset (if all data is exhausted) until max\_steps is reached.
* **warmup\_ratio** (float, *optional*, defaults to 0.0) — Ratio of total training steps used for a linear warmup from 0 to learning\_rate.
* **warmup\_steps** (int, *optional*, defaults to 0) — Number of steps used for a linear warmup from 0 to learning\_rate. Overrides any effect of warmup\_ratio.

A method that regroups all arguments linked to the learning rate scheduler and its hyperparameters.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_lr\_scheduler(name="cosine", warmup\_ratio=0.05)

>>> args.warmup\_ratio

0.05

set\_optimizer

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2553)

( name: typing.Union[str, transformers.training\_args.OptimizerNames] = 'adamw\_torch'learning\_rate: float = 5e-05weight\_decay: float = 0beta1: float = 0.9beta2: float = 0.999epsilon: float = 1e-08args: typing.Optional[str] = None )

**Parameters**

* **name** (str or training\_args.OptimizerNames, *optional*, defaults to "adamw\_torch") — The optimizer to use: "adamw\_hf", "adamw\_torch", "adamw\_torch\_fused", "adamw\_apex\_fused", "adamw\_anyprecision" or "adafactor".
* **learning\_rate** (float, *optional*, defaults to 5e-5) — The initial learning rate.
* **weight\_decay** (float, *optional*, defaults to 0) — The weight decay to apply (if not zero) to all layers except all bias and LayerNorm weights.
* **beta1** (float, *optional*, defaults to 0.9) — The beta1 hyperparameter for the adam optimizer or its variants.
* **beta2** (float, *optional*, defaults to 0.999) — The beta2 hyperparameter for the adam optimizer or its variants.
* **epsilon** (float, *optional*, defaults to 1e-8) — The epsilon hyperparameter for the adam optimizer or its variants.
* **args** (str, *optional*) — Optional arguments that are supplied to AnyPrecisionAdamW (only useful when optim="adamw\_anyprecision").

A method that regroups all arguments linked to the optimizer and its hyperparameters.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_optimizer(name="adamw\_torch", beta1=0.8)

>>> args.optim

'adamw\_torch'

set\_push\_to\_hub

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2483)

( model\_id: strstrategy: typing.Union[str, transformers.trainer\_utils.HubStrategy] = 'every\_save'token: typing.Optional[str] = Noneprivate\_repo: bool = Falsealways\_push: bool = False )

Expand 5 parameters

**Parameters**

* **model\_id** (str) — The name of the repository to keep in sync with the local *output\_dir*. It can be a simple model ID in which case the model will be pushed in your namespace. Otherwise it should be the whole repository name, for instance "user\_name/model", which allows you to push to an organization you are a member of with "organization\_name/model".
* **strategy** (str or HubStrategy, *optional*, defaults to "every\_save") — Defines the scope of what is pushed to the Hub and when. Possible values are:
  + "end": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card when the [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) method is called.
  + "every\_save": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card each time there is a model save. The pushes are asynchronous to not block training, and in case the save are very frequent, a new push is only attempted if the previous one is finished. A last push is made with the final model at the end of training.
  + "checkpoint": like "every\_save" but the latest checkpoint is also pushed in a subfolder named last-checkpoint, allowing you to resume training easily with trainer.train(resume\_from\_checkpoint="last-checkpoint").
  + "all\_checkpoints": like "checkpoint" but all checkpoints are pushed like they appear in the output folder (so you will get one checkpoint folder per folder in your final repository)
* **token** (str, *optional*) — The token to use to push the model to the Hub. Will default to the token in the cache folder obtained with huggingface-cli login.
* **private\_repo** (bool, *optional*, defaults to False) — If True, the Hub repo will be set to private.
* **always\_push** (bool, *optional*, defaults to False) — Unless this is True, the Trainer will skip pushing a checkpoint when the previous push is not finished.

A method that regroups all arguments linked to synchronizing checkpoints with the Hub.

Calling this method will set self.push\_to\_hub to True, which means the output\_dir will begin a git directory synced with the repo (determined by model\_id) and the content will be pushed each time a save is triggered (depending onself.save\_strategy). Calling [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) will also trigger a push.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_push\_to\_hub("me/awesome-model")

>>> args.hub\_model\_id

'me/awesome-model'

set\_save

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2359)

( strategy: typing.Union[str, transformers.trainer\_utils.IntervalStrategy] = 'steps'steps: int = 500total\_limit: typing.Optional[int] = Noneon\_each\_node: bool = False )

**Parameters**

* **strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The checkpoint save strategy to adopt during training. Possible values are:
  + "no": No save is done during training.
  + "epoch": Save is done at the end of each epoch.
  + "steps": Save is done every save\_steps.
* **steps** (int, *optional*, defaults to 500) — Number of updates steps before two checkpoint saves if strategy="steps".
* **total\_limit** (int, *optional*) — If a value is passed, will limit the total amount of checkpoints. Deletes the older checkpoints in output\_dir.
* **on\_each\_node** (bool, *optional*, defaults to False) — When doing multi-node distributed training, whether to save models and checkpoints on each node, or only on the main one.

This should not be activated when the different nodes use the same storage as the files will be saved with the same names for each node.

A method that regroups all arguments linked to checkpoint saving.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_save(strategy="steps", steps=100)

>>> args.save\_steps

100

set\_testing

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2319)

( batch\_size: int = 8loss\_only: bool = Falsejit\_mode: bool = False )

**Parameters**

* **batch\_size** (int *optional*, defaults to 8) — The batch size per device (GPU/TPU core/CPU…) used for testing.
* **loss\_only** (bool, *optional*, defaults to False) — Ignores all outputs except the loss.
* **jit\_mode** (bool, *optional*) — Whether or not to use PyTorch jit trace for inference.

A method that regroups all basic arguments linked to testing on a held-out dataset.

Calling this method will automatically set self.do\_predict to True.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_testing(batch\_size=32)

>>> args.per\_device\_eval\_batch\_size

32

set\_training

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2183)

( learning\_rate: float = 5e-05batch\_size: int = 8weight\_decay: float = 0num\_epochs: float = 3max\_steps: int = -1gradient\_accumulation\_steps: int = 1seed: int = 42gradient\_checkpointing: bool = False )

Expand 8 parameters

**Parameters**

* **learning\_rate** (float, *optional*, defaults to 5e-5) — The initial learning rate for the optimizer.
* **batch\_size** (int *optional*, defaults to 8) — The batch size per device (GPU/TPU core/CPU…) used for training.
* **weight\_decay** (float, *optional*, defaults to 0) — The weight decay to apply (if not zero) to all layers except all bias and LayerNorm weights in the optimizer.
* **num\_train\_epochs(**float**,** *optional*, defaults to 3.0) — Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).
* **max\_steps** (int, *optional*, defaults to -1) — If set to a positive number, the total number of training steps to perform. Overrides num\_train\_epochs. For a finite dataset, training is reiterated through the dataset (if all data is exhausted) until max\_steps is reached.
* **gradient\_accumulation\_steps** (int, *optional*, defaults to 1) — Number of updates steps to accumulate the gradients for, before performing a backward/update pass.

When using gradient accumulation, one step is counted as one step with backward pass. Therefore, logging, evaluation, save will be conducted every gradient\_accumulation\_steps \* xxx\_step training examples.

* **seed** (int, *optional*, defaults to 42) — Random seed that will be set at the beginning of training. To ensure reproducibility across runs, use the ~Trainer.model\_init function to instantiate the model if it has some randomly initialized parameters.
* **gradient\_checkpointing** (bool, *optional*, defaults to False) — If True, use gradient checkpointing to save memory at the expense of slower backward pass.

A method that regroups all basic arguments linked to the training.

Calling this method will automatically set self.do\_train to True.

Example:

Copied

>>> from transformers import TrainingArguments

>>> args = TrainingArguments("working\_dir")

>>> args = args.set\_training(learning\_rate=1e-4, batch\_size=32)

>>> args.learning\_rate

1e-4

to\_dict

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2146)

( )

Serializes this instance while replace Enum by their values (for JSON serialization support). It obfuscates the token values by removing their value.

to\_json\_string

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2163)

( )

Serializes this instance to a JSON string.

to\_sanitized\_dict

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args.py#L2169)

( )

Sanitized serialization to use with TensorBoard’s hparams

**Seq2SeqTrainingArguments**

classtransformers.**Seq2SeqTrainingArguments**

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args_seq2seq.py#L30)

( output\_dir: stroverwrite\_output\_dir: bool = Falsedo\_train: bool = Falsedo\_eval: bool = Falsedo\_predict: bool = Falseevaluation\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'no'prediction\_loss\_only: bool = Falseper\_device\_train\_batch\_size: int = 8per\_device\_eval\_batch\_size: int = 8per\_gpu\_train\_batch\_size: typing.Optional[int] = Noneper\_gpu\_eval\_batch\_size: typing.Optional[int] = Nonegradient\_accumulation\_steps: int = 1eval\_accumulation\_steps: typing.Optional[int] = Noneeval\_delay: typing.Optional[float] = 0learning\_rate: float = 5e-05weight\_decay: float = 0.0adam\_beta1: float = 0.9adam\_beta2: float = 0.999adam\_epsilon: float = 1e-08max\_grad\_norm: float = 1.0num\_train\_epochs: float = 3.0max\_steps: int = -1lr\_scheduler\_type: typing.Union[transformers.trainer\_utils.SchedulerType, str] = 'linear'lr\_scheduler\_kwargs: typing.Optional[typing.Dict] = <factory>warmup\_ratio: float = 0.0warmup\_steps: int = 0log\_level: typing.Optional[str] = 'passive'log\_level\_replica: typing.Optional[str] = 'warning'log\_on\_each\_node: bool = Truelogging\_dir: typing.Optional[str] = Nonelogging\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'steps'logging\_first\_step: bool = Falselogging\_steps: float = 500logging\_nan\_inf\_filter: bool = Truesave\_strategy: typing.Union[transformers.trainer\_utils.IntervalStrategy, str] = 'steps'save\_steps: float = 500save\_total\_limit: typing.Optional[int] = Nonesave\_safetensors: typing.Optional[bool] = Truesave\_on\_each\_node: bool = Falsesave\_only\_model: bool = Falseno\_cuda: bool = Falseuse\_cpu: bool = Falseuse\_mps\_device: bool = Falseseed: int = 42data\_seed: typing.Optional[int] = Nonejit\_mode\_eval: bool = Falseuse\_ipex: bool = Falsebf16: bool = Falsefp16: bool = Falsefp16\_opt\_level: str = 'O1'half\_precision\_backend: str = 'auto'bf16\_full\_eval: bool = Falsefp16\_full\_eval: bool = Falsetf32: typing.Optional[bool] = Nonelocal\_rank: int = -1ddp\_backend: typing.Optional[str] = Nonetpu\_num\_cores: typing.Optional[int] = Nonetpu\_metrics\_debug: bool = Falsedebug: typing.Union[str, typing.List[transformers.debug\_utils.DebugOption]] = ''dataloader\_drop\_last: bool = Falseeval\_steps: typing.Optional[float] = Nonedataloader\_num\_workers: int = 0past\_index: int = -1run\_name: typing.Optional[str] = Nonedisable\_tqdm: typing.Optional[bool] = Noneremove\_unused\_columns: typing.Optional[bool] = Truelabel\_names: typing.Optional[typing.List[str]] = Noneload\_best\_model\_at\_end: typing.Optional[bool] = Falsemetric\_for\_best\_model: typing.Optional[str] = Nonegreater\_is\_better: typing.Optional[bool] = Noneignore\_data\_skip: bool = Falsefsdp: typing.Union[typing.List[transformers.trainer\_utils.FSDPOption], str, NoneType] = ''fsdp\_min\_num\_params: int = 0fsdp\_config: typing.Optional[str] = Nonefsdp\_transformer\_layer\_cls\_to\_wrap: typing.Optional[str] = Nonedeepspeed: typing.Optional[str] = Nonelabel\_smoothing\_factor: float = 0.0optim: typing.Union[transformers.training\_args.OptimizerNames, str] = 'adamw\_torch'optim\_args: typing.Optional[str] = Noneadafactor: bool = Falsegroup\_by\_length: bool = Falselength\_column\_name: typing.Optional[str] = 'length'report\_to: typing.Optional[typing.List[str]] = Noneddp\_find\_unused\_parameters: typing.Optional[bool] = Noneddp\_bucket\_cap\_mb: typing.Optional[int] = Noneddp\_broadcast\_buffers: typing.Optional[bool] = Nonedataloader\_pin\_memory: bool = Truedataloader\_persistent\_workers: bool = Falseskip\_memory\_metrics: bool = Trueuse\_legacy\_prediction\_loop: bool = Falsepush\_to\_hub: bool = Falseresume\_from\_checkpoint: typing.Optional[str] = Nonehub\_model\_id: typing.Optional[str] = Nonehub\_strategy: typing.Union[transformers.trainer\_utils.HubStrategy, str] = 'every\_save'hub\_token: typing.Optional[str] = Nonehub\_private\_repo: bool = Falsehub\_always\_push: bool = Falsegradient\_checkpointing: bool = Falsegradient\_checkpointing\_kwargs: typing.Optional[dict] = Noneinclude\_inputs\_for\_metrics: bool = Falsefp16\_backend: str = 'auto'push\_to\_hub\_model\_id: typing.Optional[str] = Nonepush\_to\_hub\_organization: typing.Optional[str] = Nonepush\_to\_hub\_token: typing.Optional[str] = Nonemp\_parameters: str = ''auto\_find\_batch\_size: bool = Falsefull\_determinism: bool = Falsetorchdynamo: typing.Optional[str] = Noneray\_scope: typing.Optional[str] = 'last'ddp\_timeout: typing.Optional[int] = 1800torch\_compile: bool = Falsetorch\_compile\_backend: typing.Optional[str] = Nonetorch\_compile\_mode: typing.Optional[str] = Nonedispatch\_batches: typing.Optional[bool] = Nonesplit\_batches: typing.Optional[bool] = Falseinclude\_tokens\_per\_second: typing.Optional[bool] = Falseinclude\_num\_input\_tokens\_seen: typing.Optional[bool] = Falseneftune\_noise\_alpha: float = Nonesortish\_sampler: bool = Falsepredict\_with\_generate: bool = Falsegeneration\_max\_length: typing.Optional[int] = Nonegeneration\_num\_beams: typing.Optional[int] = Nonegeneration\_config: typing.Union[str, pathlib.Path, transformers.generation.configuration\_utils.GenerationConfig, NoneType] = None )

Expand 110 parameters

**Parameters**

* **output\_dir** (str) — The output directory where the model predictions and checkpoints will be written.
* **overwrite\_output\_dir** (bool, *optional*, defaults to False) — If True, overwrite the content of the output directory. Use this to continue training if output\_dir points to a checkpoint directory.
* **do\_train** (bool, *optional*, defaults to False) — Whether to run training or not. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **do\_eval** (bool, *optional*) — Whether to run evaluation on the validation set or not. Will be set to True if evaluation\_strategy is different from "no". This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **do\_predict** (bool, *optional*, defaults to False) — Whether to run predictions on the test set or not. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **evaluation\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "no") — The evaluation strategy to adopt during training. Possible values are:
  + "no": No evaluation is done during training.
  + "steps": Evaluation is done (and logged) every eval\_steps.
  + "epoch": Evaluation is done at the end of each epoch.
* **prediction\_loss\_only** (bool, *optional*, defaults to False) — When performing evaluation and generating predictions, only returns the loss.
* **per\_device\_train\_batch\_size** (int, *optional*, defaults to 8) — The batch size per GPU/XPU/TPU/MPS/NPU core/CPU for training.
* **per\_device\_eval\_batch\_size** (int, *optional*, defaults to 8) — The batch size per GPU/XPU/TPU/MPS/NPU core/CPU for evaluation.
* **gradient\_accumulation\_steps** (int, *optional*, defaults to 1) — Number of updates steps to accumulate the gradients for, before performing a backward/update pass.

When using gradient accumulation, one step is counted as one step with backward pass. Therefore, logging, evaluation, save will be conducted every gradient\_accumulation\_steps \* xxx\_step training examples.

* **eval\_accumulation\_steps** (int, *optional*) — Number of predictions steps to accumulate the output tensors for, before moving the results to the CPU. If left unset, the whole predictions are accumulated on GPU/NPU/TPU before being moved to the CPU (faster but requires more memory).
* **eval\_delay** (float, *optional*) — Number of epochs or steps to wait for before the first evaluation can be performed, depending on the evaluation\_strategy.
* **learning\_rate** (float, *optional*, defaults to 5e-5) — The initial learning rate for [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **weight\_decay** (float, *optional*, defaults to 0) — The weight decay to apply (if not zero) to all layers except all bias and LayerNorm weights in [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_beta1** (float, *optional*, defaults to 0.9) — The beta1 hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_beta2** (float, *optional*, defaults to 0.999) — The beta2 hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **adam\_epsilon** (float, *optional*, defaults to 1e-8) — The epsilon hyperparameter for the [AdamW](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.AdamW) optimizer.
* **max\_grad\_norm** (float, *optional*, defaults to 1.0) — Maximum gradient norm (for gradient clipping).
* **num\_train\_epochs(**float**,** *optional*, defaults to 3.0) — Total number of training epochs to perform (if not an integer, will perform the decimal part percents of the last epoch before stopping training).
* **max\_steps** (int, *optional*, defaults to -1) — If set to a positive number, the total number of training steps to perform. Overrides num\_train\_epochs. For a finite dataset, training is reiterated through the dataset (if all data is exhausted) until max\_steps is reached.
* **lr\_scheduler\_type** (str or [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType), *optional*, defaults to "linear") — The scheduler type to use. See the documentation of [SchedulerType](https://huggingface.co/docs/transformers/main/en/main_classes/optimizer_schedules#transformers.SchedulerType) for all possible values.
* **lr\_scheduler\_kwargs** (‘dict’, *optional*, defaults to {}) — The extra arguments for the lr\_scheduler. See the documentation of each scheduler for possible values.
* **warmup\_ratio** (float, *optional*, defaults to 0.0) — Ratio of total training steps used for a linear warmup from 0 to learning\_rate.
* **warmup\_steps** (int, *optional*, defaults to 0) — Number of steps used for a linear warmup from 0 to learning\_rate. Overrides any effect of warmup\_ratio.
* **log\_level** (str, *optional*, defaults to passive) — Logger log level to use on the main process. Possible choices are the log levels as strings: ‘debug’, ‘info’, ‘warning’, ‘error’ and ‘critical’, plus a ‘passive’ level which doesn’t set anything and keeps the current log level for the Transformers library (which will be "warning" by default).
* **log\_level\_replica** (str, *optional*, defaults to "warning") — Logger log level to use on replicas. Same choices as log\_level”
* **log\_on\_each\_node** (bool, *optional*, defaults to True) — In multinode distributed training, whether to log using log\_level once per node, or only on the main node.
* **logging\_dir** (str, *optional*) — [TensorBoard](https://www.tensorflow.org/tensorboard) log directory. Will default to \*output\_dir/runs/**CURRENT\_DATETIME\_HOSTNAME\***.
* **logging\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The logging strategy to adopt during training. Possible values are:
  + "no": No logging is done during training.
  + "epoch": Logging is done at the end of each epoch.
  + "steps": Logging is done every logging\_steps.
* **logging\_first\_step** (bool, *optional*, defaults to False) — Whether to log and evaluate the first global\_step or not.
* **logging\_steps** (int or float, *optional*, defaults to 500) — Number of update steps between two logs if logging\_strategy="steps". Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **logging\_nan\_inf\_filter** (bool, *optional*, defaults to True) — Whether to filter nan and inf losses for logging. If set to True the loss of every step that is nan or inf is filtered and the average loss of the current logging window is taken instead.

logging\_nan\_inf\_filter only influences the logging of loss values, it does not change the behavior the gradient is computed or applied to the model.

* **save\_strategy** (str or [IntervalStrategy](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.IntervalStrategy), *optional*, defaults to "steps") — The checkpoint save strategy to adopt during training. Possible values are:
  + "no": No save is done during training.
  + "epoch": Save is done at the end of each epoch.
  + "steps": Save is done every save\_steps.
* **save\_steps** (int or float, *optional*, defaults to 500) — Number of updates steps before two checkpoint saves if save\_strategy="steps". Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **save\_total\_limit** (int, *optional*) — If a value is passed, will limit the total amount of checkpoints. Deletes the older checkpoints in output\_dir. When load\_best\_model\_at\_end is enabled, the “best” checkpoint according to metric\_for\_best\_model will always be retained in addition to the most recent ones. For example, for save\_total\_limit=5 and load\_best\_model\_at\_end, the four last checkpoints will always be retained alongside the best model. When save\_total\_limit=1 and load\_best\_model\_at\_end, it is possible that two checkpoints are saved: the last one and the best one (if they are different).
* **save\_safetensors** (bool, *optional*, defaults to True) — Use [safetensors](https://huggingface.co/docs/safetensors) saving and loading for state dicts instead of default torch.load and torch.save.
* **save\_on\_each\_node** (bool, *optional*, defaults to False) — When doing multi-node distributed training, whether to save models and checkpoints on each node, or only on the main one.

This should not be activated when the different nodes use the same storage as the files will be saved with the same names for each node.

* **save\_only\_model** (bool, *optional*, defaults to False) — When checkpointing, whether to only save the model, or also the optimizer, scheduler & rng state. Note that when this is true, you won’t be able to resume training from checkpoint. This enables you to save storage by not storing the optimizer, scheduler & rng state. You can only load the model using from\_pretrained with this option set to True.
* **use\_cpu** (bool, *optional*, defaults to False) — Whether or not to use cpu. If set to False, we will use cuda or mps device if available.
* **seed** (int, *optional*, defaults to 42) — Random seed that will be set at the beginning of training. To ensure reproducibility across runs, use the ~Trainer.model\_init function to instantiate the model if it has some randomly initialized parameters.
* **data\_seed** (int, *optional*) — Random seed to be used with data samplers. If not set, random generators for data sampling will use the same seed as seed. This can be used to ensure reproducibility of data sampling, independent of the model seed.
* **jit\_mode\_eval** (bool, *optional*, defaults to False) — Whether or not to use PyTorch jit trace for inference.
* **use\_ipex** (bool, *optional*, defaults to False) — Use Intel extension for PyTorch when it is available. [IPEX installation](https://github.com/intel/intel-extension-for-pytorch).
* **bf16** (bool, *optional*, defaults to False) — Whether to use bf16 16-bit (mixed) precision training instead of 32-bit training. Requires Ampere or higher NVIDIA architecture or using CPU (use\_cpu) or Ascend NPU. This is an experimental API and it may change.
* **fp16** (bool, *optional*, defaults to False) — Whether to use fp16 16-bit (mixed) precision training instead of 32-bit training.
* **fp16\_opt\_level** (str, *optional*, defaults to ‘O1’) — For fp16 training, Apex AMP optimization level selected in [‘O0’, ‘O1’, ‘O2’, and ‘O3’]. See details on the [Apex documentation](https://nvidia.github.io/apex/amp).
* **fp16\_backend** (str, *optional*, defaults to "auto") — This argument is deprecated. Use half\_precision\_backend instead.
* **half\_precision\_backend** (str, *optional*, defaults to "auto") — The backend to use for mixed precision training. Must be one of "auto", "apex", "cpu\_amp". "auto" will use CPU/CUDA AMP or APEX depending on the PyTorch version detected, while the other choices will force the requested backend.
* **bf16\_full\_eval** (bool, *optional*, defaults to False) — Whether to use full bfloat16 evaluation instead of 32-bit. This will be faster and save memory but can harm metric values. This is an experimental API and it may change.
* **fp16\_full\_eval** (bool, *optional*, defaults to False) — Whether to use full float16 evaluation instead of 32-bit. This will be faster and save memory but can harm metric values.
* **tf32** (bool, *optional*) — Whether to enable the TF32 mode, available in Ampere and newer GPU architectures. The default value depends on PyTorch’s version default of torch.backends.cuda.matmul.allow\_tf32. For more details please refer to the [TF32](https://huggingface.co/docs/transformers/performance#tf32) documentation. This is an experimental API and it may change.
* **local\_rank** (int, *optional*, defaults to -1) — Rank of the process during distributed training.
* **ddp\_backend** (str, *optional*) — The backend to use for distributed training. Must be one of "nccl", "mpi", "ccl", "gloo", "hccl".
* **tpu\_num\_cores** (int, *optional*) — When training on TPU, the number of TPU cores (automatically passed by launcher script).
* **dataloader\_drop\_last** (bool, *optional*, defaults to False) — Whether to drop the last incomplete batch (if the length of the dataset is not divisible by the batch size) or not.
* **eval\_steps** (int or float, *optional*) — Number of update steps between two evaluations if evaluation\_strategy="steps". Will default to the same value as logging\_steps if not set. Should be an integer or a float in range [0,1). If smaller than 1, will be interpreted as ratio of total training steps.
* **dataloader\_num\_workers** (int, *optional*, defaults to 0) — Number of subprocesses to use for data loading (PyTorch only). 0 means that the data will be loaded in the main process.
* **past\_index** (int, *optional*, defaults to -1) — Some models like [TransformerXL](https://huggingface.co/docs/transformers/main/model_doc/transformerxl) or [XLNet](https://huggingface.co/docs/transformers/main/model_doc/xlnet) can make use of the past hidden states for their predictions. If this argument is set to a positive int, the Trainer will use the corresponding output (usually index 2) as the past state and feed it to the model at the next training step under the keyword argument mems.
* **run\_name** (str, *optional*) — A descriptor for the run. Typically used for [wandb](https://www.wandb.com/) and [mlflow](https://www.mlflow.org/) logging.
* **disable\_tqdm** (bool, *optional*) — Whether or not to disable the tqdm progress bars and table of metrics produced by ~notebook.NotebookTrainingTracker in Jupyter Notebooks. Will default to True if the logging level is set to warn or lower (default), False otherwise.
* **remove\_unused\_columns** (bool, *optional*, defaults to True) — Whether or not to automatically remove the columns unused by the model forward method.

(Note that this behavior is not implemented for TFTrainer yet.)

* **label\_names** (List[str], *optional*) — The list of keys in your dictionary of inputs that correspond to the labels.

Will eventually default to the list of argument names accepted by the model that contain the word “label”, except if the model used is one of the XxxForQuestionAnswering in which case it will also include the ["start\_positions", "end\_positions"] keys.

* **load\_best\_model\_at\_end** (bool, *optional*, defaults to False) — Whether or not to load the best model found during training at the end of training. When this option is enabled, the best checkpoint will always be saved. See [save\_total\_limit](https://huggingface.co/docs/transformers/main_classes/trainer#transformers.TrainingArguments.save_total_limit) for more.

When set to True, the parameters save\_strategy needs to be the same as evaluation\_strategy, and in the case it is “steps”, save\_steps must be a round multiple of eval\_steps.

* **metric\_for\_best\_model** (str, *optional*) — Use in conjunction with load\_best\_model\_at\_end to specify the metric to use to compare two different models. Must be the name of a metric returned by the evaluation with or without the prefix "eval\_". Will default to "loss" if unspecified and load\_best\_model\_at\_end=True (to use the evaluation loss).

If you set this value, greater\_is\_better will default to True. Don’t forget to set it to False if your metric is better when lower.

* **greater\_is\_better** (bool, *optional*) — Use in conjunction with load\_best\_model\_at\_end and metric\_for\_best\_model to specify if better models should have a greater metric or not. Will default to:
  + True if metric\_for\_best\_model is set to a value that isn’t "loss" or "eval\_loss".
  + False if metric\_for\_best\_model is not set, or set to "loss" or "eval\_loss".
* **ignore\_data\_skip** (bool, *optional*, defaults to False) — When resuming training, whether or not to skip the epochs and batches to get the data loading at the same stage as in the previous training. If set to True, the training will begin faster (as that skipping step can take a long time) but will not yield the same results as the interrupted training would have.
* **fsdp** (bool, str or list of FSDPOption, *optional*, defaults to '') — Use PyTorch Distributed Parallel Training (in distributed training only).

A list of options along the following:

* + "full\_shard": Shard parameters, gradients and optimizer states.
  + "shard\_grad\_op": Shard optimizer states and gradients.
  + "hybrid\_shard": Apply FULL\_SHARD within a node, and replicate parameters across nodes.
  + "hybrid\_shard\_zero2": Apply SHARD\_GRAD\_OP within a node, and replicate parameters across nodes.
  + "offload": Offload parameters and gradients to CPUs (only compatible with "full\_shard" and "shard\_grad\_op").
  + "auto\_wrap": Automatically recursively wrap layers with FSDP using default\_auto\_wrap\_policy.
* **fsdp\_config** (str or dict, *optional*) — Config to be used with fsdp (Pytorch Distributed Parallel Training). The value is either a location of fsdp json config file (e.g., fsdp\_config.json) or an already loaded json file as dict.

A List of config and its options:

* + min\_num\_params (int, *optional*, defaults to 0): FSDP’s minimum number of parameters for Default Auto Wrapping. (useful only when fsdp field is passed).
  + transformer\_layer\_cls\_to\_wrap (List[str], *optional*): List of transformer layer class names (case-sensitive) to wrap, e.g, BertLayer, GPTJBlock, T5Block … (useful only when fsdp flag is passed).
  + backward\_prefetch (str, *optional*) FSDP’s backward prefetch mode. Controls when to prefetch next set of parameters (useful only when fsdp field is passed).

A list of options along the following:

* + - "backward\_pre" : Prefetches the next set of parameters before the current set of parameter’s gradient computation.
    - "backward\_post" : This prefetches the next set of parameters after the current set of parameter’s gradient computation.
  + forward\_prefetch (bool, *optional*, defaults to False) FSDP’s forward prefetch mode (useful only when fsdp field is passed). If "True", then FSDP explicitly prefetches the next upcoming all-gather while executing in the forward pass.
  + limit\_all\_gathers (bool, *optional*, defaults to False) FSDP’s limit\_all\_gathers (useful only when fsdp field is passed). If "True", FSDP explicitly synchronizes the CPU thread to prevent too many in-flight all-gathers.
  + use\_orig\_params (bool, *optional*, defaults to True) If "True", allows non-uniform requires\_grad during init, which means support for interspersed frozen and trainable paramteres. Useful in cases such as parameter-efficient fine-tuning. Please refer this [blog](<https://dev-discuss.pytorch.org/t/rethinking-pytorch-fully-sharded-data-parallel-fsdp-from-first-principles/1019>
  + sync\_module\_states (bool, *optional*, defaults to True) If "True", each individually wrapped FSDP unit will broadcast module parameters from rank 0 to ensure they are the same across all ranks after initialization
  + activation\_checkpointing (bool, *optional*, defaults to False): If "True", activation checkpointing is a technique to reduce memory usage by clearing activations of certain layers and recomputing them during a backward pass. Effectively, this trades extra computation time for reduced memory usage.
  + xla (bool, *optional*, defaults to False): Whether to use PyTorch/XLA Fully Sharded Data Parallel Training. This is an experimental feature and its API may evolve in the future.
  + xla\_fsdp\_settings (dict, *optional*) The value is a dictionary which stores the XLA FSDP wrapping parameters.

For a complete list of options, please see [here](https://github.com/pytorch/xla/blob/master/torch_xla/distributed/fsdp/xla_fully_sharded_data_parallel.py).

* + xla\_fsdp\_grad\_ckpt (bool, *optional*, defaults to False): Will use gradient checkpointing over each nested XLA FSDP wrapped layer. This setting can only be used when the xla flag is set to true, and an auto wrapping policy is specified through fsdp\_min\_num\_params or fsdp\_transformer\_layer\_cls\_to\_wrap.
* **deepspeed** (str or dict, *optional*) — Use [Deepspeed](https://github.com/microsoft/deepspeed). This is an experimental feature and its API may evolve in the future. The value is either the location of DeepSpeed json config file (e.g., ds\_config.json) or an already loaded json file as a dict”
* **label\_smoothing\_factor** (float, *optional*, defaults to 0.0) — The label smoothing factor to use. Zero means no label smoothing, otherwise the underlying onehot-encoded labels are changed from 0s and 1s to label\_smoothing\_factor/num\_labels and 1 - label\_smoothing\_factor + label\_smoothing\_factor/num\_labels respectively.
* **debug** (str or list of DebugOption, *optional*, defaults to "") — Enable one or more debug features. This is an experimental feature.

Possible options are:

* + "underflow\_overflow": detects overflow in model’s input/outputs and reports the last frames that led to the event
  + "tpu\_metrics\_debug": print debug metrics on TPU

The options should be separated by whitespaces.

* **optim** (str or training\_args.OptimizerNames, *optional*, defaults to "adamw\_torch") — The optimizer to use: adamw\_hf, adamw\_torch, adamw\_torch\_fused, adamw\_apex\_fused, adamw\_anyprecision or adafactor.
* **optim\_args** (str, *optional*) — Optional arguments that are supplied to AnyPrecisionAdamW.
* **group\_by\_length** (bool, *optional*, defaults to False) — Whether or not to group together samples of roughly the same length in the training dataset (to minimize padding applied and be more efficient). Only useful if applying dynamic padding.
* **length\_column\_name** (str, *optional*, defaults to "length") — Column name for precomputed lengths. If the column exists, grouping by length will use these values rather than computing them on train startup. Ignored unless group\_by\_length is True and the dataset is an instance of Dataset.
* **report\_to** (str or List[str], *optional*, defaults to "all") — The list of integrations to report the results and logs to. Supported platforms are "azure\_ml", "clearml", "codecarbon", "comet\_ml", "dagshub", "dvclive", "flyte", "mlflow", "neptune", "tensorboard", and "wandb". Use "all" to report to all integrations installed, "none" for no integrations.
* **ddp\_find\_unused\_parameters** (bool, *optional*) — When using distributed training, the value of the flag find\_unused\_parameters passed to DistributedDataParallel. Will default to False if gradient checkpointing is used, True otherwise.
* **ddp\_bucket\_cap\_mb** (int, *optional*) — When using distributed training, the value of the flag bucket\_cap\_mb passed to DistributedDataParallel.
* **ddp\_broadcast\_buffers** (bool, *optional*) — When using distributed training, the value of the flag broadcast\_buffers passed to DistributedDataParallel. Will default to False if gradient checkpointing is used, True otherwise.
* **dataloader\_pin\_memory** (bool, *optional*, defaults to True) — Whether you want to pin memory in data loaders or not. Will default to True.
* **dataloader\_persistent\_workers** (bool, *optional*, defaults to False) — If True, the data loader will not shut down the worker processes after a dataset has been consumed once. This allows to maintain the workers Dataset instances alive. Can potentially speed up training, but will increase RAM usage. Will default to False.
* **skip\_memory\_metrics** (bool, *optional*, defaults to True) — Whether to skip adding of memory profiler reports to metrics. This is skipped by default because it slows down the training and evaluation speed.
* **push\_to\_hub** (bool, *optional*, defaults to False) — Whether or not to push the model to the Hub every time the model is saved. If this is activated, output\_dir will begin a git directory synced with the repo (determined by hub\_model\_id) and the content will be pushed each time a save is triggered (depending on your save\_strategy). Calling [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) will also trigger a push.

If output\_dir exists, it needs to be a local clone of the repository to which the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer) will be pushed.

* **resume\_from\_checkpoint** (str, *optional*) — The path to a folder with a valid checkpoint for your model. This argument is not directly used by [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer), it’s intended to be used by your training/evaluation scripts instead. See the [example scripts](https://github.com/huggingface/transformers/tree/main/examples) for more details.
* **hub\_model\_id** (str, *optional*) — The name of the repository to keep in sync with the local *output\_dir*. It can be a simple model ID in which case the model will be pushed in your namespace. Otherwise it should be the whole repository name, for instance "user\_name/model", which allows you to push to an organization you are a member of with "organization\_name/model". Will default to user\_name/output\_dir\_name with *output\_dir\_name* being the name of output\_dir.

Will default to the name of output\_dir.

* **hub\_strategy** (str or HubStrategy, *optional*, defaults to "every\_save") — Defines the scope of what is pushed to the Hub and when. Possible values are:
  + "end": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card when the [save\_model()](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer.save_model) method is called.
  + "every\_save": push the model, its configuration, the tokenizer (if passed along to the [Trainer](https://huggingface.co/docs/transformers/main/en/main_classes/trainer#transformers.Trainer)) and a draft of a model card each time there is a model save. The pushes are asynchronous to not block training, and in case the save are very frequent, a new push is only attempted if the previous one is finished. A last push is made with the final model at the end of training.
  + "checkpoint": like "every\_save" but the latest checkpoint is also pushed in a subfolder named last-checkpoint, allowing you to resume training easily with trainer.train(resume\_from\_checkpoint="last-checkpoint").
  + "all\_checkpoints": like "checkpoint" but all checkpoints are pushed like they appear in the output folder (so you will get one checkpoint folder per folder in your final repository)
* **hub\_token** (str, *optional*) — The token to use to push the model to the Hub. Will default to the token in the cache folder obtained with huggingface-cli login.
* **hub\_private\_repo** (bool, *optional*, defaults to False) — If True, the Hub repo will be set to private.
* **hub\_always\_push** (bool, *optional*, defaults to False) — Unless this is True, the Trainer will skip pushing a checkpoint when the previous push is not finished.
* **gradient\_checkpointing** (bool, *optional*, defaults to False) — If True, use gradient checkpointing to save memory at the expense of slower backward pass.
* **gradient\_checkpointing\_kwargs** (dict, *optional*, defaults to None) — Key word arguments to be passed to the gradient\_checkpointing\_enable method.
* **include\_inputs\_for\_metrics** (bool, *optional*, defaults to False) — Whether or not the inputs will be passed to the compute\_metrics function. This is intended for metrics that need inputs, predictions and references for scoring calculation in Metric class.
* **auto\_find\_batch\_size** (bool, *optional*, defaults to False) — Whether to find a batch size that will fit into memory automatically through exponential decay, avoiding CUDA Out-of-Memory errors. Requires accelerate to be installed (pip install accelerate)
* **full\_determinism** (bool, *optional*, defaults to False) — If True, [enable\_full\_determinism()](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.enable_full_determinism) is called instead of [set\_seed()](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.set_seed) to ensure reproducible results in distributed training. Important: this will negatively impact the performance, so only use it for debugging.
* **torchdynamo** (str, *optional*) — If set, the backend compiler for TorchDynamo. Possible choices are "eager", "aot\_eager", "inductor", "nvfuser", "aot\_nvfuser", "aot\_cudagraphs", "ofi", "fx2trt", "onnxrt" and "ipex".
* **ray\_scope** (str, *optional*, defaults to "last") — The scope to use when doing hyperparameter search with Ray. By default, "last" will be used. Ray will then use the last checkpoint of all trials, compare those, and select the best one. However, other options are also available. See the [Ray documentation](https://docs.ray.io/en/latest/tune/api_docs/analysis.html#ray.tune.ExperimentAnalysis.get_best_trial) for more options.
* **ddp\_timeout** (int, *optional*, defaults to 1800) — The timeout for torch.distributed.init\_process\_group calls, used to avoid GPU socket timeouts when performing slow operations in distributed runnings. Please refer the [PyTorch documentation] (<https://pytorch.org/docs/stable/distributed.html#torch.distributed.init_process_group>) for more information.
* **use\_mps\_device** (bool, *optional*, defaults to False) — This argument is deprecated.mps device will be used if it is available similar to cuda device.
* **torch\_compile** (bool, *optional*, defaults to False) — Whether or not to compile the model using PyTorch 2.0 [torch.compile](https://pytorch.org/get-started/pytorch-2.0/).

This will use the best defaults for the [torch.compile API](https://pytorch.org/docs/stable/generated/torch.compile.html?highlight=torch+compile#torch.compile). You can customize the defaults with the argument torch\_compile\_backend and torch\_compile\_mode but we don’t guarantee any of them will work as the support is progressively rolled in in PyTorch.

This flag and the whole compile API is experimental and subject to change in future releases.

* **torch\_compile\_backend** (str, *optional*) — The backend to use in torch.compile. If set to any value, torch\_compile will be set to True.

Refer to the PyTorch doc for possible values and note that they may change across PyTorch versions.

This flag is experimental and subject to change in future releases.

* **torch\_compile\_mode** (str, *optional*) — The mode to use in torch.compile. If set to any value, torch\_compile will be set to True.

Refer to the PyTorch doc for possible values and note that they may change across PyTorch versions.

This flag is experimental and subject to change in future releases.

* **split\_batches** (bool, *optional*) — Whether or not the accelerator should split the batches yielded by the dataloaders across the devices during distributed training. If

set to True, the actual batch size used will be the same on any kind of distributed processes, but it must be a

round multiple of the number of processes you are using (such as GPUs).

* **include\_tokens\_per\_second** (bool, *optional*) — Whether or not to compute the number of tokens per second per device for training speed metrics.

This will iterate over the entire training dataloader once beforehand,

and will slow down the entire process.

* **include\_num\_input\_tokens\_seen** (bool, *optional*) — Whether or not to track the number of input tokens seen throughout training.

May be slower in distributed training as gather operations must be called.

* **neftune\_noise\_alpha** (Optional[float]) — If not None, this will activate NEFTune noise embeddings. This can drastically improve model performance for instruction fine-tuning. Check out the [original paper](https://arxiv.org/abs/2310.05914) and the [original code](https://github.com/neelsjain/NEFTune). Support transformers PreTrainedModel and also PeftModel from peft.
* **sortish\_sampler** (bool, *optional*, defaults to False) — Whether to use a *sortish sampler* or not. Only possible if the underlying datasets are *Seq2SeqDataset* for now but will become generally available in the near future.

It sorts the inputs according to lengths in order to minimize the padding size, with a bit of randomness for the training set.

* **predict\_with\_generate** (bool, *optional*, defaults to False) — Whether to use generate to calculate generative metrics (ROUGE, BLEU).
* **generation\_max\_length** (int, *optional*) — The max\_length to use on each evaluation loop when predict\_with\_generate=True. Will default to the max\_length value of the model configuration.
* **generation\_num\_beams** (int, *optional*) — The num\_beams to use on each evaluation loop when predict\_with\_generate=True. Will default to the num\_beams value of the model configuration.
* **generation\_config** (str or Path or [GenerationConfig](https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.GenerationConfig), *optional*) — Allows to load a [GenerationConfig](https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.GenerationConfig) from the from\_pretrained method. This can be either:
  + a string, the *model id* of a pretrained model configuration hosted inside a model repo on huggingface.co. Valid model ids can be located at the root-level, like bert-base-uncased, or namespaced under a user or organization name, like dbmdz/bert-base-german-cased.
  + a path to a *directory* containing a configuration file saved using the [save\_pretrained()](https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.GenerationConfig.save_pretrained) method, e.g., ./my\_model\_directory/.
  + a [GenerationConfig](https://huggingface.co/docs/transformers/main/en/main_classes/text_generation#transformers.GenerationConfig) object.

TrainingArguments is the subset of the arguments we use in our example scripts **which relate to the training loop itself**.

Using [HfArgumentParser](https://huggingface.co/docs/transformers/main/en/internal/trainer_utils#transformers.HfArgumentParser) we can turn this class into [argparse](https://docs.python.org/3/library/argparse#module-argparse) arguments that can be specified on the command line.

to\_dict

[<source>](https://github.com/huggingface/transformers/blob/main/src/transformers/training_args_seq2seq.py#L87)

( )

Serializes this instance while replace Enum by their values and GenerationConfig by dictionaries (for JSON serialization support). It obfuscates the token values by removing their value.