**Reward Modeling**

TRL supports custom reward modeling for anyone to perform reward modeling on their dataset and model.

Check out a complete flexible example inside [examples/scripts](https://github.com/huggingface/trl/tree/main/examples/scripts/reward_modeling.py) folder.

**Expected dataset format**

The [RewardTrainer](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardTrainer) expects a very specific format for the dataset since the model will be trained on pairs of examples to predict which of the two is preferred. We provide an example from the [Anthropic/hh-rlhf](https://huggingface.co/datasets/Anthropic/hh-rlhf) dataset below:

Therefore the final dataset object should contain two 4 entries at least if you use the default RewardDataCollatorWithPadding data collator. The entries should be named:

* input\_ids\_chosen
* attention\_mask\_chosen
* input\_ids\_rejected
* attention\_mask\_rejected

**Using the RewardTrainer**

After preparing your dataset, you can use the [RewardTrainer](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardTrainer) in the same way as the Trainer class from 🤗 Transformers. You should pass an AutoModelForSequenceClassification model to the [RewardTrainer](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardTrainer), along with a [RewardConfig](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardConfig) which configures the hyperparameters of the training.

**Leveraging 🤗 PEFT to train a reward model**

Just pass a peft\_config in the keyword arguments of [RewardTrainer](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardTrainer), and the trainer should automatically take care of converting the model into a PEFT model!

Copied

from peft import LoraConfig, task\_type

from transformers import AutoModelForSequenceClassification, AutoTokenizer

from trl import RewardTrainer, RewardConfig

model = AutoModelForSequenceClassification.from\_pretrained("gpt2")

peft\_config = LoraConfig(

task\_type=TaskType.SEQ\_CLS,

inference\_mode=False,

r=8,

lora\_alpha=32,

lora\_dropout=0.1,

)

...

trainer = RewardTrainer(

model=model,

args=training\_args,

tokenizer=tokenizer,

train\_dataset=dataset,

peft\_config=peft\_config,

)

trainer.train()

**Adding a margin to the loss**

As in the [Llama 2 paper](https://huggingface.co/papers/2307.09288), you can add a margin to the loss by adding a margin column to the dataset. The reward collator will automatically pass it through and the loss will be computed accordingly.

Copied

def add\_margin(row):

*# Assume you have a score\_chosen and score\_rejected columns that you want to use to compute the margin*

return {'margin': row['score\_chosen'] - row['score\_rejected']}

dataset = dataset.map(add\_margin)

**RewardConfig**

classtrl.**RewardConfig**

[<source>](https://github.com/huggingface/trl/blob/v0.7.4/trl/trainer/training_configs.py#L23)

( 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 = 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 = Trueskip\_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: typing.Optional[bool] = Truegradient\_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] = Falseneftune\_noise\_alpha: float = Nonemax\_length: typing.Optional[int] = None )

**Parameters**

* **max\_length** (int, *optional*, defaults to None) — The maximum length of the sequences in the batch. This argument is required if you want to use the default data collator.
* **gradient\_checkpointing** (bool, *optional*, defaults to True) — If True, use gradient checkpointing to save memory at the expense of slower backward pass.

RewardConfig collects all training arguments related to the [RewardTrainer](https://huggingface.co/docs/trl/v0.7.4/en/reward_trainer" \l "trl.RewardTrainer) class.

Using HfArgumentParser we can turn this class into [argparse](https://docs.python.org/3/library/argparse" \l "module-argparse) arguments that can be specified on the command line.

**RewardTrainer**

classtrl.**RewardTrainer**

[<source>](https://github.com/huggingface/trl/blob/v0.7.4/trl/trainer/reward_trainer.py#L36)

( model: typing.Union[transformers.modeling\_utils.PreTrainedModel, torch.nn.modules.module.Module] = Noneargs: typing.Optional[trl.trainer.training\_configs.RewardConfig] = Nonedata\_collator: typing.Optional[DataCollator] = Nonetrain\_dataset: typing.Optional[datasets.arrow\_dataset.Dataset] = Noneeval\_dataset: typing.Union[datasets.arrow\_dataset.Dataset, typing.Dict[str, datasets.arrow\_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] = Nonemax\_length: typing.Optional[int] = Nonepeft\_config: typing.Optional[typing.Dict] = None )

The RewardTrainer can be used to train your custom Reward Model. It is a subclass of the transformers.Trainer class and inherits all of its attributes and methods. It is recommended to use an AutoModelForSequenceClassification as the reward model. The reward model should be trained on a dataset of paired examples, where each example is a tuple of two sequences. The reward model should be trained to predict which example in the pair is more relevant to the task at hand.

The reward trainer expects a very specific format for the dataset. The dataset should contain two 4 entries at least if you don’t use the default RewardDataCollatorWithPadding data collator. The entries should be named

* input\_ids\_chosen
* attention\_mask\_chosen
* input\_ids\_rejected
* attention\_mask\_rejected

Optionally, you can also pass a margin entry to the dataset. This entry should contain the margin used to modulate the loss of the reward model as outlined in <https://ai.meta.com/research/publications/llama-2-open-foundation-and-fine-tuned-chat-models/>. If you don’t pass a margin, no margin will be used.