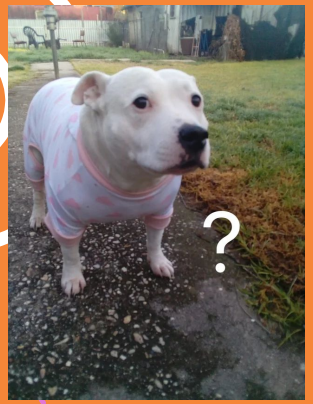
Fine Tuning with TRL

Sergio Paniego Blanco (@sergiopaniego)

Machine Learning Engineer @ Hugging Face

From fine-tuning to TRL (fine-tuning+alignment)



“I fine-tuned the model and …” 😅

💬 It can’t chat naturally
🧠 It can’t reason or explain
🌀 It gives incoherent answers
🔁 It gives inconsistent answers
⚠️ It’s not aligned with human preferences (unhelpful or unsafe)
🚫 It doesn’t follow instructions
…

From fine-tuning to TRL (fine-tuning+alignment)

Classic fine-tuning adapts the model to **your dataset**, not necessarily to **your intentions**.

- Dataset (X→Y) → Pretrained model → Fine-tuned model
- Optimizes likelihood: predicts correctly based on the data
- **Doesn't guarantee alignment** with human preferences

We may need something stronger (TRL !), tools that can teach the model, not just fit it (TRL ?!).

Data → Model → Fine-tuned model
💭 (Still says weird stuff)

Where does TRL fits in the model lifecycle? 🤔

Pre-training: teaches general capacities such as language use, broad reasoning, and world knowledge
Mid-training: imparts domain knowledge (code, medical databases, internal docs…)
Post-training/Alignment (TRL!): learns to behave as we want (instruction following, reasoning…)

TRL is **not for pretraining** → it's for the final stages, where we make models useful, safe, and aligned

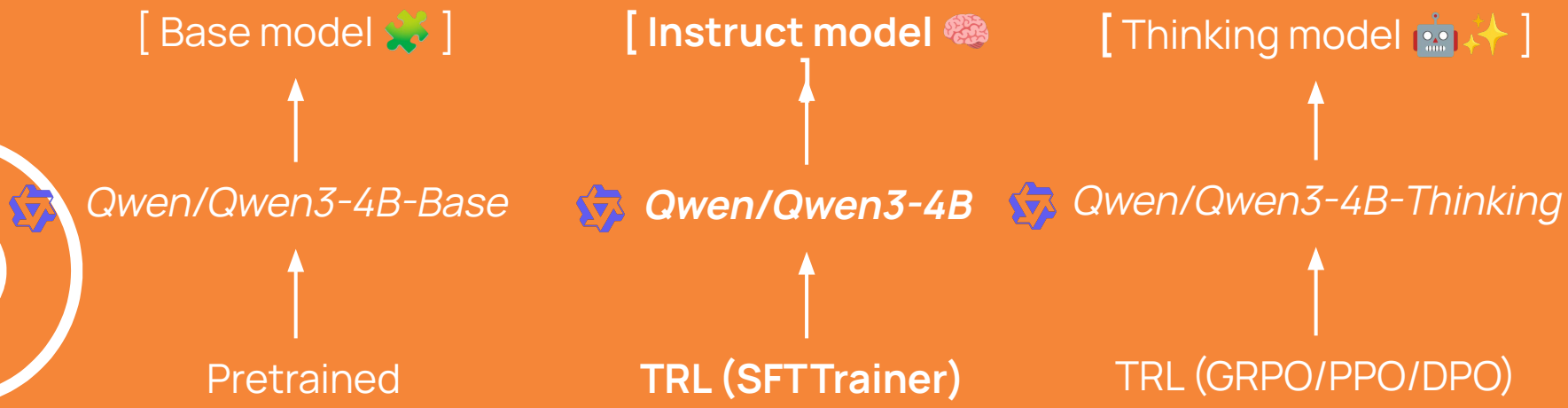
From token prediction to alignment

[Base model 🧩]	[Instruct model 🧠]	[Thinking model 🤖✨]
Qwen/Qwen3-4B-Base	*Qwen/Qwen3-4B*	*Qwen/Qwen3-4B-Thinking*
Pretrained	TRL (SFTTrainer)	TRL (GRPO/PPO/DPO)

Base: predicts next token
Raw text, high quantity of data, masked language modelling, industrial compute

"*Translate this sentence → Translate this sentence into...*"

From token prediction to alignment



Instruct: follows instructions
Structured responses, high quality data, reduced compute

"‹|user|› *Translate this sentence* ‹/s›
‹|assistant|› *Sure! Here's the translation* ‹/s›"

Example dataset

From token prediction to alignment

[Base model 🧩] → *Qwen/Qwen3-4B-Base* → Pretrained

[Instruct model 🧠] → *Qwen/Qwen3-4B* → TRL (SFTTrainer)

[**Thinking model** 🤖✨] → ***Qwen/Qwen3-4B-Thinking*** → **TRL (**GRPO/**PPO/DPO)**

Aligning model: helpfulness, truthfulness, and safety.

Preferences, high quality, preference optim, consumer compute

"‹|user|› *How many helicopters can a human eat?‹/s›*

👎 ‹*|assistant|› Humans cannot eat helicopters‹/s›*

👍 ‹*|assistant|› That's not possible!‹/s›*"

Example dataset

From token prediction to alignment

[Base model 🧩] → *Qwen/Qwen3-4B-Base* → Pretrained

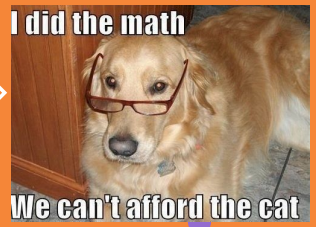
[Instruct model 🧠] → *Qwen/Qwen3-4B* → TRL (SFTTrainer)

[**Thinking model** 🤖✨] → ***Qwen/Qwen3-4B-Thinking*** → **TRL (GRPO**/PPO/DPO**)**

Thinking model: reasoning!

- “‹|user|› *How many helicopters can a human eat?‹/s›*
  ‹|assistant|›
  ‹think› “Tricky question. Let’s break it down…‹/think›
  That’s not possible! I found that…‹/s›”

- Example dataset



TRL: Fine-tuning + alignment!

TRL: HF library that offers specialized trainers (SFT, GRPO, PPO, DPO…)

- TRL bridges the gap between **instruction following** and **full alignment pipelines**.
- Compatible with **accelerate, PEFT, kernels, trackio, vLLM…**
- **Multimodal** models support…
- **Post-training: SFT, alignment (DPO), reasoning (GRPO)**

Trainer 🆚 TRL

TRL trainers are **built on top of transformers Trainer**, enhancing its functionality.
Each trainer **abstracts a full training method**, so you don't need to reimplement everything from scratch.
Gives ML engineers **ready-to-use recipes** for SFT, GRPO, PPO, DPO...
Focus: practicality, reproducibility, and reduced boilerplate

```python
from trl import SFTTrainer
from datasets import load_dataset

trainer = SFTTrainer(
    model="Qwen/Qwen3-0.6B",
    train_dataset=load_dataset("trl-lib/Capybara", split="train"),
)
trainer.train()
```

SFT and GRPO

We'll focus here on two trainers. But there are more 👀

Supervised Fine-Tuning (SFT)

- Teach the model to follow instructions
  Uses human-labeled data (prompt → response)
- Improves helpfulness

Group Relative Policy Optimization (GRPO)

- Train with online reinforcement learning
- Uses reward models or human preference signals
- Improves alignment and reasoning

TRL taxonomy

Online methods

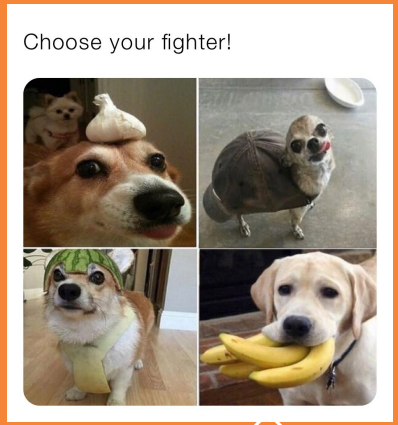
(learn from feedback) Model generates reponses and learns from reward signals during training

- **GRPO** ⚡
- RLOO ⚡
- OnlineDPO ⚡
- NashMD ⚡
- XPO ⚡
- PPO

Offline methods

(data-driven) Model learns from human preference datasets or supervised signals.

- **SFT**
- DPO
- ORPO
- BCO
- CPO
- KTO



Reward modeling

Training a model to judge responses

- PRM
- Reward

Knowledge distillation

Transferring knowledge from a stronger model to a smaller one

- GKD

⚡ = vLLM **support**

SFT and GRPO are all you need?

SFT and GRPO dominate the scene (apparently)
TRL includes +10 trainers → research moves fast!
Hard to maintain? There's even an open issue in the repo
New training methods appear!

[RFC] Moving Most TRL Trainers to the experimental Submodule to Streamline the Core #4223

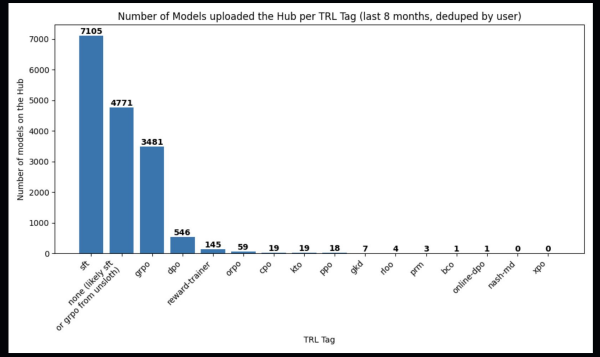
Open

qgallouedec opened 3 weeks ago · edited by qgallouedec

Edits · Member

Context

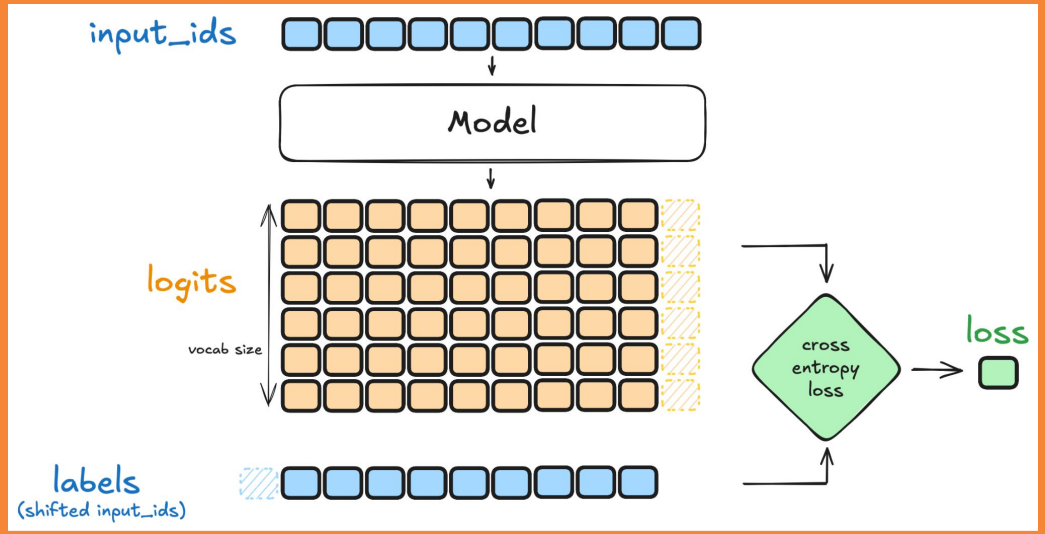
TRL currently includes 15 trainers, which vary significantly in usage and maintenance requirements.



Supervised Fine-Tuning

SFT is the simplest and most commonly used method to adapt a LM to a target dataset

- **Goal:** minimize negative log-likelihood of a sequence
- Simple, stable, and super effective.



Supervised Fine-Tuning

SFTTrainer supports both **standard** and **conversational** dataset formats.

```
# Standard
language_modeling_example = {"text": "The sky is blue."}

# Conversational
messages = [
    {"role": "user", "content": "Hello, how are you?"},
    {"role": "assistant", "content": "I'm doing great. How can I help you today?"},
    {"role": "user", "content": "I'd like to show off how chat templating works!"},
]
```

SFT is easy to instantiate and train

Only needs a model (even just the name!) and a dataset from the Hub to train

```python
from trl import SFTTrainer
from datasets import load_dataset

trainer = SFTTrainer(
    model="Qwen/Qwen3-0.6B",
    train_dataset=load_dataset("trl-lib/Capybara", split="train"),
)
trainer.train()
```

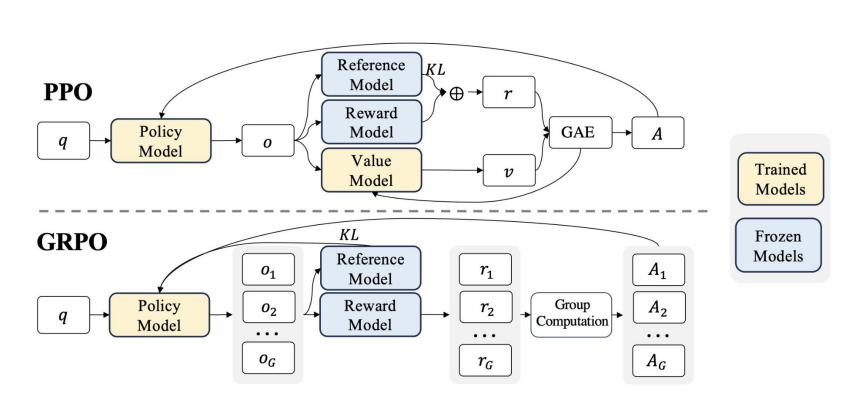
SFTTrainer and SFTConfig

```python
from trl import SFTTrainer, SFTConfig
from datasets import load_dataset

trainer = SFTTrainer(
    model="Qwen/Qwen3-0.6B",
    train_dataset=load_dataset("trl-lib/Capybara", split="train"),
    args = SFTConfig(
      # Training schedule / optimization
      per_device_train_batch_size = 2,      # Batch size per GPU
      gradient_accumulation_steps = 8,      # Gradients are accumulated over multiple steps → effective batch size = 2 * 8 = 16
      num_train_epochs = 1,                # Number of full dataset passes
      learning_rate = 2e-4,                 # Learning rate for the optimizer
      optim = "paged_adamw_8bit",           # Optimizer

      # Logging / reporting
      logging_steps=1,                      # Log training metrics every N steps
      report_to="trackio",                  # Experiment tracking tool

      # Hub integration
      push_to_hub=True,                     # Automatically push the trained model to the Hugging Face Hub
      ...
    )
    peft_config=LoraConfig(),
    # ...
)
trainer.train()
```

Group Relative Policy Optimization

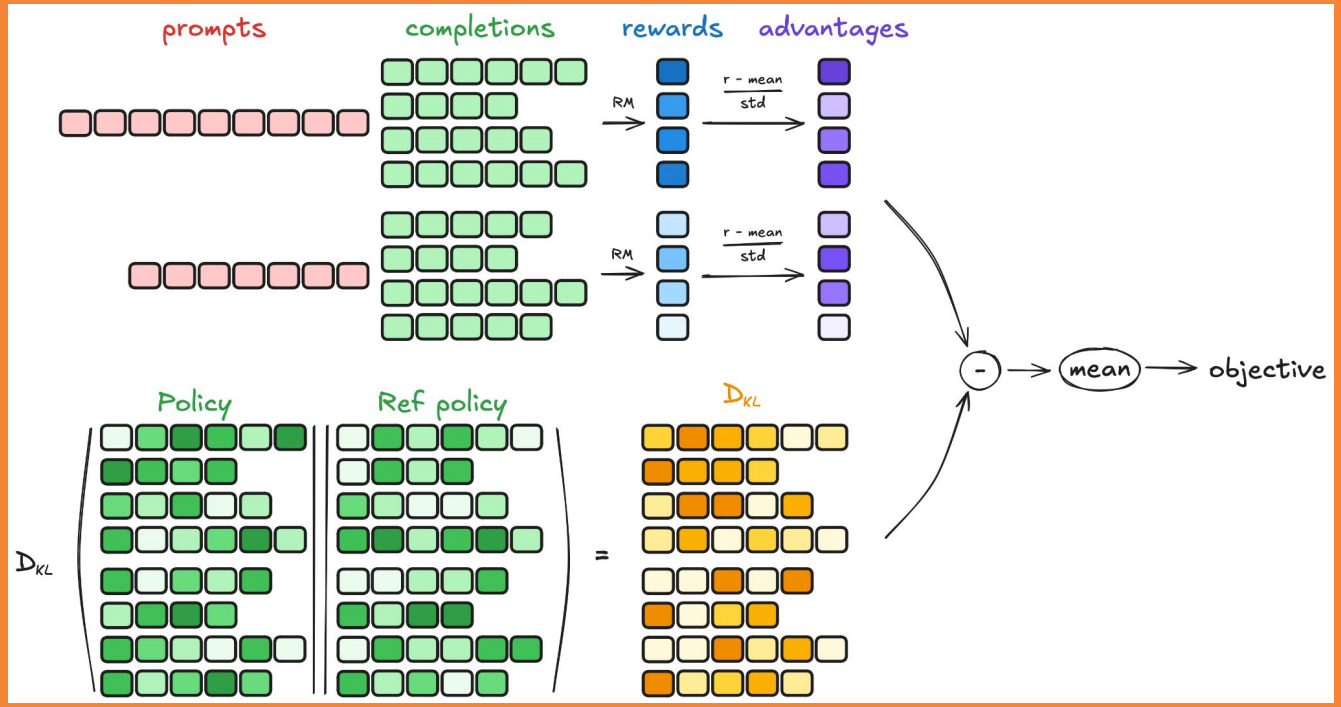
GRPO was described in the DeepSeek paper.
Variant of Proximal Policy Optimization (PPO), that enhances maths reasoning abilities while optimizing memory usage

- **Online** method ⚡
- Optimizes behavior using **rewards** from **preference models** or heuristics (**reward functions**)
- Successor of PPO: more stable and memory-efficient



GRPO

Steps: generating completions, computing advantage, ~~estimating the KL divergence~~, computing the loss



GRPOTrainer and GRPOConfig

```python
from trl import GRPOTrainer, GRPOConfig
import re

def format_reward_func(completions, **kwargs):
    """Reward function that checks if the completion has a specific format."""
    pattern = r"^<think>.*?</think><answer>.*?</answer>$"
    completion_contents = [completion[0]["content"] for completion in completions]
    matches = [re.match(pattern, content) for content in completion_contents]
    return [1.0 if match else 0.0 for match in matches]

trainer = GRPOTrainer(
    model="Qwen/Qwen2-0.5B-Instruct",
    reward_funcs=[format_reward_func, ...],
    args=GRPOConfig(...),
    train_dataset=load_dataset("trl-lib/tldr", split="train"),
    # ...
)
trainer.train()
```

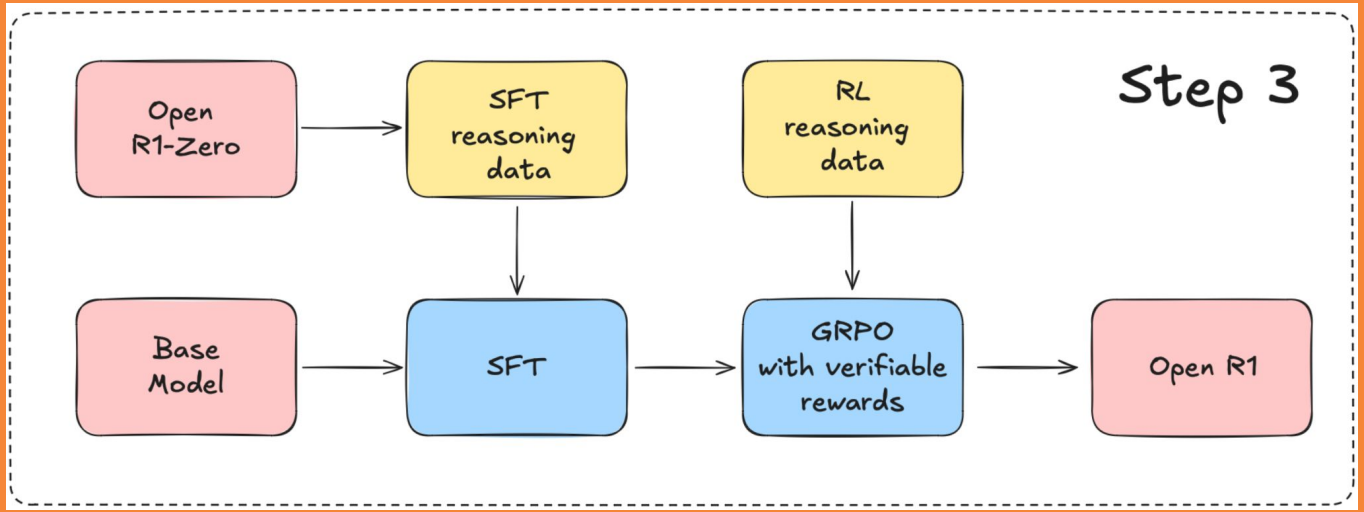
GRPOConfig detail

```python
from trl import GRPOConfig

# Configure training arguments using GRPOConfig
training_args = GRPOConfig(
    # ... Similar to SFTConfig

    # Parameters that control de data preprocessing
    max_completion_length=512,  # default: 256
    num_generations=8,  # default: 8
    max_prompt_length=512,  # default: 512

)
```

OpenR1

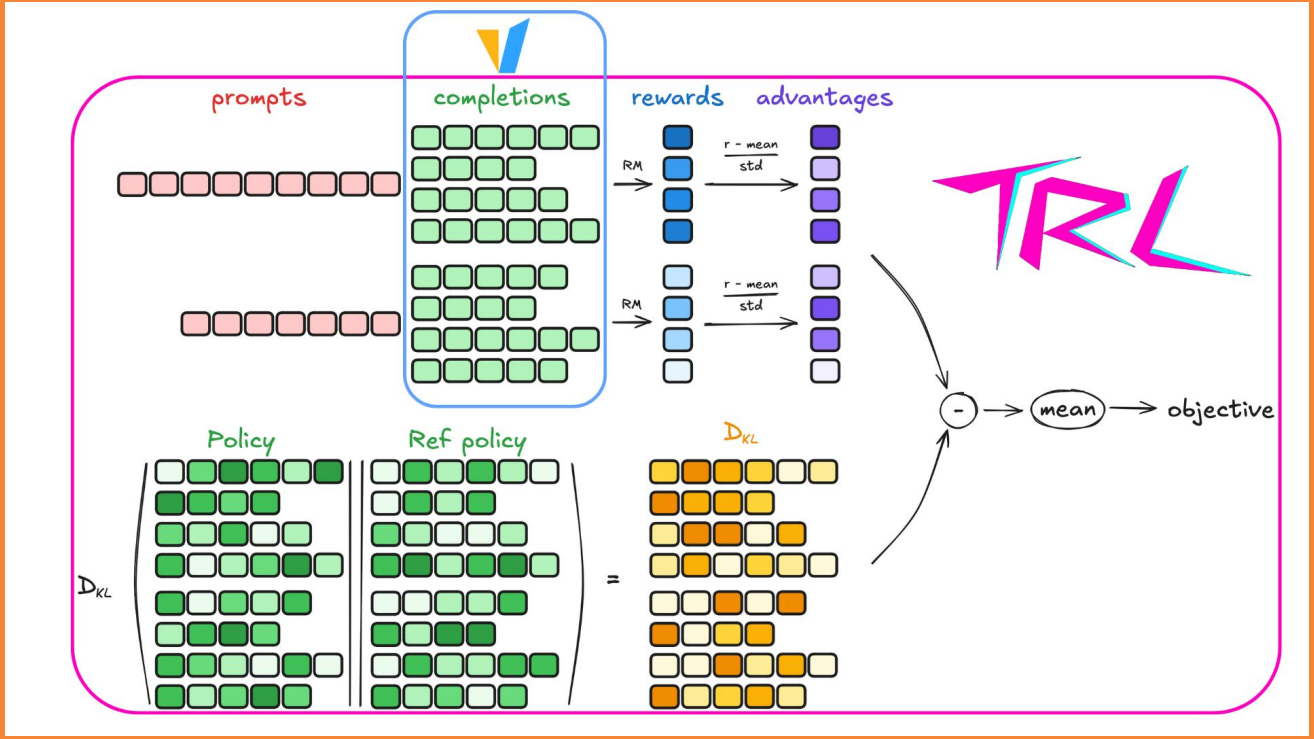
Initiative by HF to replicate and extend the techniques behind DeepSeek-R1 🐳

This model was built using SFT + GRPO!



Can we improve online training?

Some TRL trainers work online ⚡, the model generates completions during training to compute a reward signal.



Can we improve online training?

🚧 **Bottleneck!** Generation is slow and inefficient
⚡ **vLLM** solves this problem and TRL supports it!
You can call it from the **CLI** (yes, TRL has a CLI)

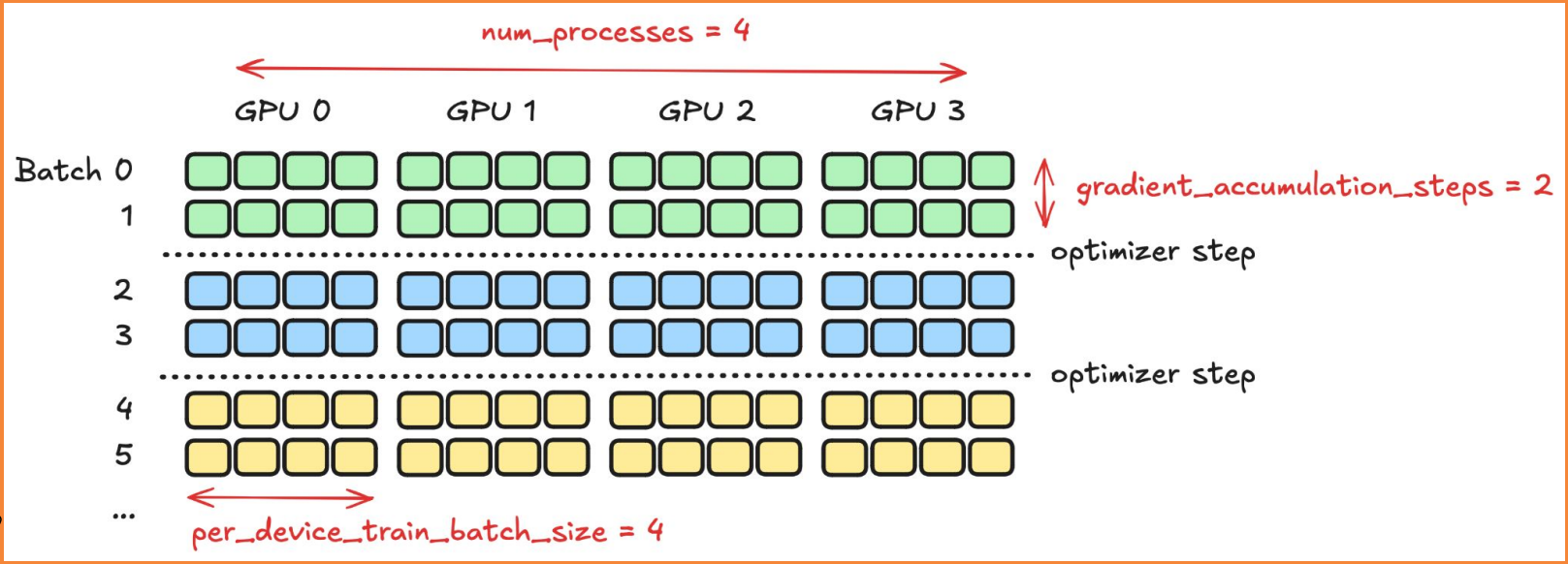

```
CUDA_VISIBLE_DEVICES=0,1,2,3 trl vllm-serve --model Qwen/Qwen2.5-7B --tensor-parallel-size 2 --data-parallel-size 2
```

```python
from trl import GRPOConfig

training_args = GRPOConfig(
    ...,
    use_vllm=True,
    vllm_mode="server",  # default value, can be omitted. Can also be "colocate"
)
```

But can we scale training?

TRL also integrates seamlessly with accelerate and DeepSpeed (model sharing, zero redundancy optimized, mixed precision training, offloading…)

deepspeed



But can we scale training?

In the config we specify num of machines, num of processes...

```
accelerate launch --config_file examples/accelerate_configs/deepspeed_zero2.yaml train.py
```

But can we scale training?

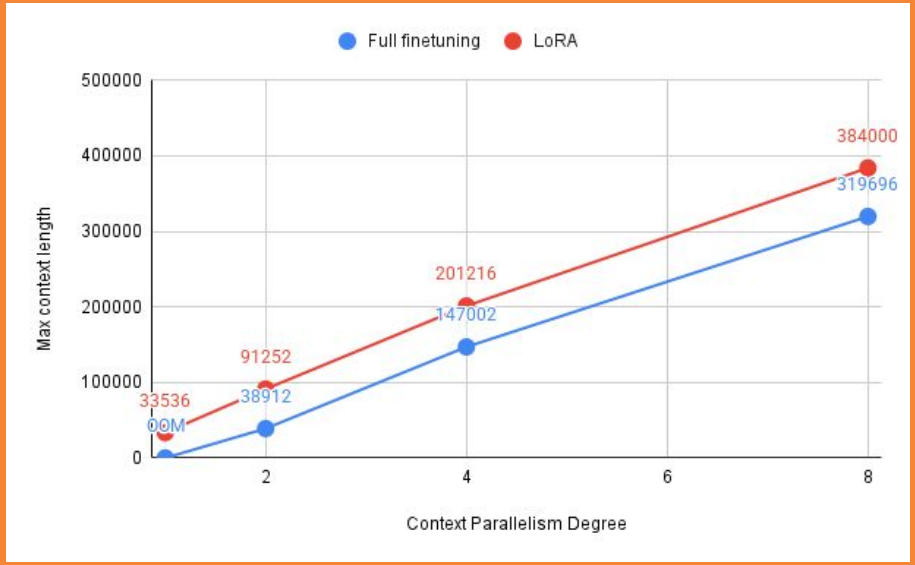
Train long contexts via **context parallelism** *(accelerate)*: by splitting the sequence across multiple GPUs
Even **ND-parallelism** (N model replicas, N devices, tensor parallelism, context parallelism)

```
accelerate launch --config_file context_parallel_2gpu.yaml train.py
```

```
from trl import SFTConfig

training_args = SFTConfig(
    # required
    pad_to_multiple_of=4,           # ensures divisibility by cp_size * 2
    # to get the most out of CP
    max_length=400000,              # long sequence length
    packing=True,                   # use packing to reduce padding
    use_liger_kernel=True,          # compatible with CP
    gradient_checkpointing=False,   # The activation_checkpointing in FSDP config and
the gradient_checkpointing in training arg can't be set to True simultaneously
    per_device_train_batch_size=1,
    ...
)
```

But can we scale training?

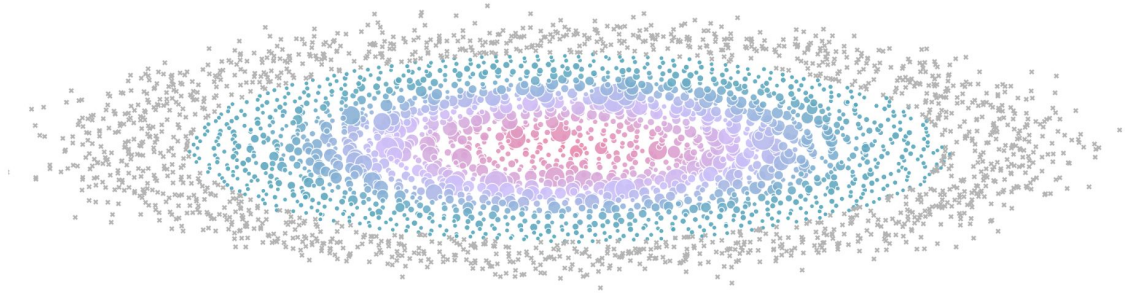
Context Parallelism(CP) with *Qwen/Qwen3-8B* scales to over **300k tokens in 8 GPUs** 🙌



But can we scale training?

nanotron / ultrascale-playbook

The Ultra-Scale Playbook: Training LLMs on GPU Clusters



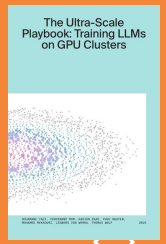
We ran over 4,000 scaling experiments on up to 512 GPUs and measured throughput (size of markers) and GPU utilization (color of markers). Note that both are normalized per model size in this visualization.

ORDER BOOK GET PDF

AUTHORS
Nouamane Tazi, Ferdinand Mom, Haojun Zhao, Phuc Nguyen, Mohamed Mekkouri, Leandro Werra, Thomas Wolf

AFFILIATION
Hugging Face

PUBLISHED
Feb 19, 2025



Optimizing all the things!

Transformers already provides us with some optimizing strategies like gradient checkpointing and we also have LoRA/QLoRA. In addition, TRL has:

- Truncation
- Packing
- Padding-free
- Padding sequences to a multiple
- Liger kernels 

- Activation offloading

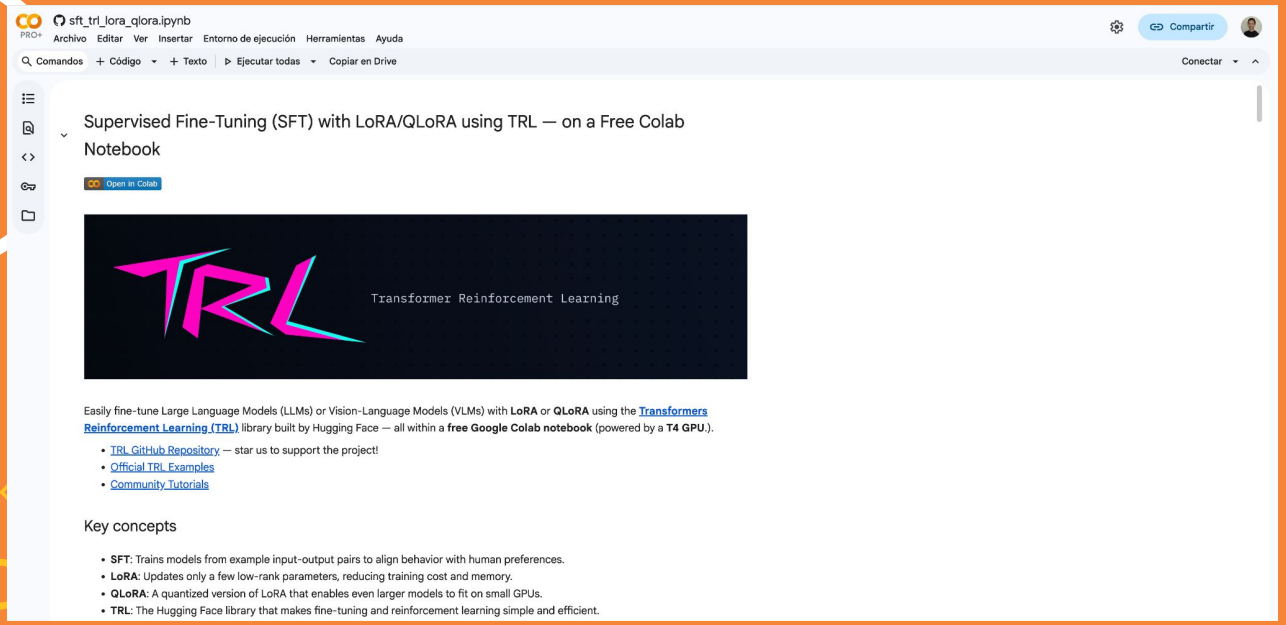
We developed a notebook for training a **14B** model using QLoRA in **free Colab**!

Optimizing is super easy

```python
from trl import SFTConfig

training_args = SFTConfig(
    max_length=1024,                        # (Truncation) Maximum input sequence length
    packing=True                            # Packing
    padding_free=True, model_init_kwargs={"attn_implementation": "flash_attention_2"}, # Padding-free
    pad_to_multiple_of=2048                 # Padding sequences to a multiple
    use_liger_kernel=True,                  # Enable Liger kernel optimizations for faster training
    activation_offloading=True,             # Offload activations to CPU to reduce GPU memory usage

    gradient_checkpointing=True,            # Save memory by re-computing activations during backpropagation
)
```

Optimizing for GPU poors

We developed a notebook for training a **14B** model using QLoRA in **free Colab**!



HF jobs + TRL

Models can be trained using HF's infra via HF Jobs

- **trl-jobs** cli package
- **hf jobs** (cli or python)

```
# pip install trl-jobs
trl-jobs sft --model_name Qwen/Qwen3-0.6B --dataset_name trl-lib/Capybara
```

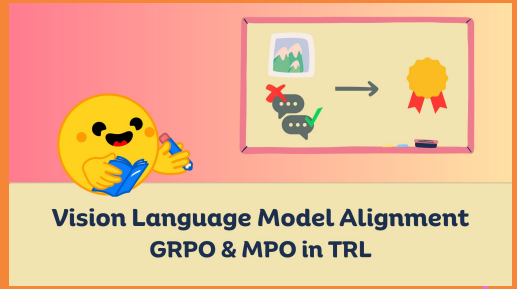
```
hf jobs uv run \
    --flavor a100-large \
    --with trl \
    --secrets HF_TOKEN \
    train.py
```

```
from huggingface_hub import run_uv_job

run_uv_job(
    "train.py",
    dependencies=["trl"],
    flavor="a100-large",
    secrets={"HF_TOKEN": "hf_..."},
)
```

Multimodality! 🖼️

Vision Language Models (**VLMs**) are getting stronger, but **aligning** them to human preferences still matters.
TRL supports VLMs in many trainers, including examples and recipes.

- SFT
- DPO (Direct Preference Optimization)



- MPO (Mixed Preference Optimization)
- GRPO (Group Relative Policy Optimization)
- GSPO (Group Sequence Policy Optimization)

- RLOO (Reinforce Leave One Out)
- Online DPO

Multimodality!! 🖼️

Native SFT support for VLMs
Just can still use your own collator

```python
from trl import SFTConfig, SFTTrainer
from datasets import load_dataset

trainer = SFTTrainer(
    model="Qwen/Qwen2.5-VL-3B-Instruct",
    args=SFTConfig(max_length=None), # To avoid truncation that may remove image tokens during training
    train_dataset=load_dataset("trl-lib/llava-instruct-mix", split="train"),
)
trainer.train()
```

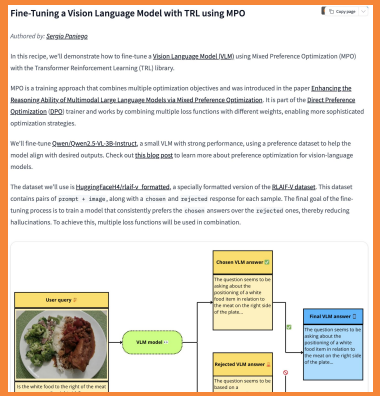
Example dataset

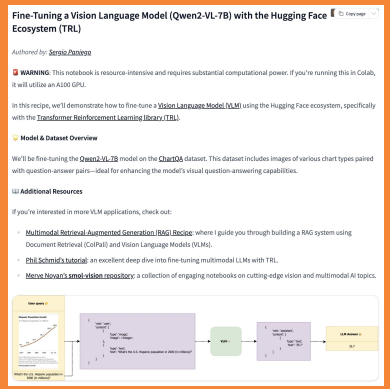
Multimodality!!!

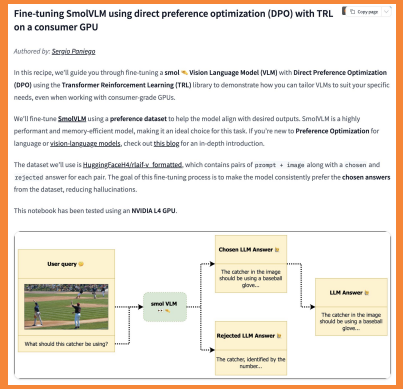
Shared scripts and recipes!

examples/scripts/dpo_vlm.py	This script shows how to use the DPOTrainer to fine-tune a Vision Language Model to reduce hallucinations using the openbmb/RLAIF-V-Dataset dataset.
examples/scripts/evals/judge_tldr.py	This script shows how to use HfPairwiseJudge or OpenAIPairwiseJudge to judge model generations.
examples/scripts/gkd.py	This script shows how to use the GKDTrainer to fine-tune a model.
trl/scripts/grpo.py	This script shows how to use the GRPOTrainer to fine-tune a model.
examples/scripts/grpo_vlm.py	This script shows how to use the GRPOTrainer to fine-tune a multimodal model for reasoning using the lmms-lab/multimodal-open-r1-8k-verified dataset.
examples/scripts/gspo.py	This script shows how to use GSPO via the GRPOTrainer to fine-tune model for reasoning using the AI-MO/NuminaMath-TIR dataset.
examples/scripts/gspo_vlm.py	This script shows how to use GSPO via the GRPOTrainer to fine-tune a multimodal model for reasoning using the lmms-lab/multimodal-open-r1-8k-verified dataset.
examples/scripts/kto.py	This script shows how to use the KTOTrainer to fine-tune a model.
examples/scripts/mpo_vlm.py	This script shows how to use MPO via the DPOTrainer to align a model based on preferences using the HuggingFaceH4/rlaif-v_formatted dataset and a set of loss weights with weights.
examples/scripts/nash_md.py	This script shows how to use the NashMDTrainer to fine-tune a model.
examples/scripts/online_dpo.py	This script shows how to use the OnlineDPOTrainer to fine-tune a model.
examples/scripts/online_dpo_vlm.py	This script shows how to use the OnlineDPOTrainer to fine-tune a a Vision Language Model.

examples/scripts/sft_vlm.py	This script shows how to use the SFTTrainer to fine-tune a Vision Language Model in a chat setting. The script has only been tested with LLaVA 1.5, LLaVA 1.6, and Llama-3.2-11B-Vision-Instruct models so users may see unexpected behaviour in other model architectures.
examples/scripts/sft_vlm_gemma3.py	This script shows how to use the SFTTrainer to fine-tune a Gemma 3 model on vision to text tasks.
examples/scripts/sft_vlm_smol_vlm.py	This script shows how to use the SFTTrainer to fine-tune a SmolVLM model.







Agents 🕵️ via OpenEnv

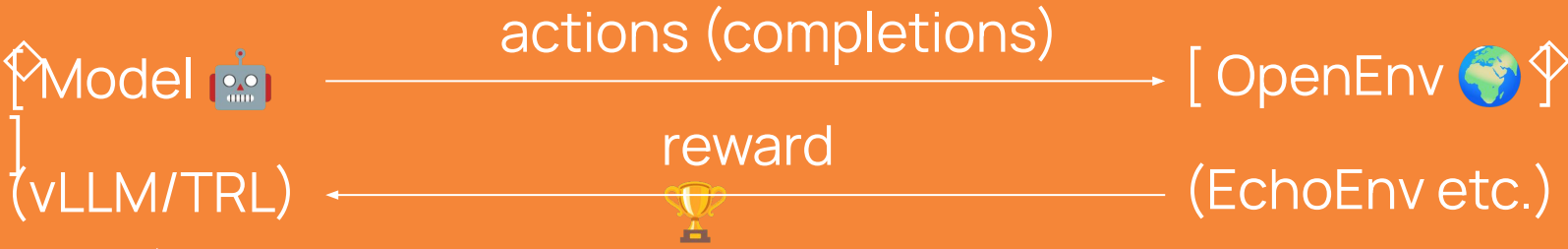
🧩 **OpenEnv** → framework for interactive TRL environments (*Gymnasium* for LLMs)

🖥️ Runs envs locally or as backend servers

🌐 Find ready-to-use OpenEnv envs on the Hub

🔗 Integrates with TRL → use `*rollout_func*` in GRPO to replace text generation with environment interaction

🎯 Models can act, observe, and learn, not just predict text.

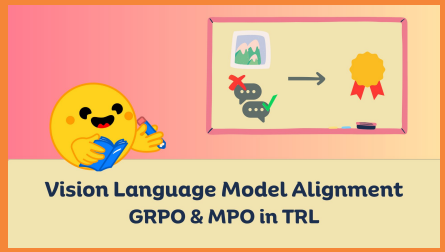


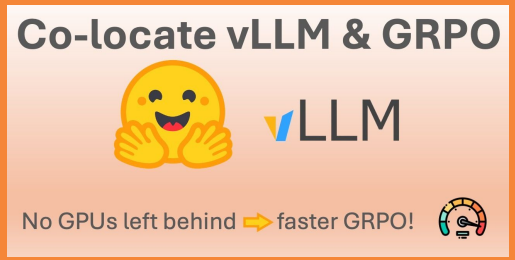
Agents 🕵️ via OpenEnv

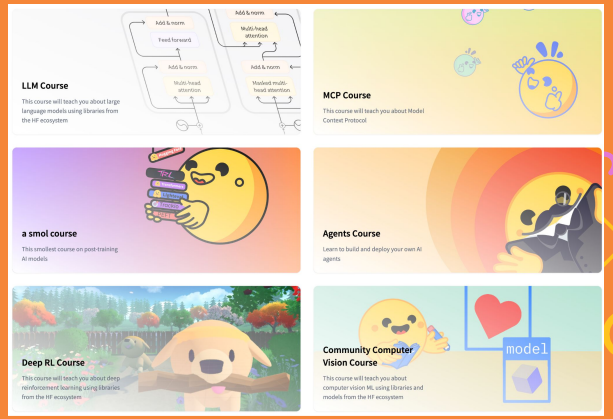
```python
from envs.echo_env import EchoEnv, EchoAction
from trl import GRPOConfig, GRPOTrainer

# Create HTTP client for Echo Environment
client = EchoEnv.from_docker_image("echo-env:latest")

def rollout_func(prompts, args, processing_class):
    # 1. Generate completions via vLLM inference server (running on port 8000)
    ...
    response = requests.post("http://0.0.0.0:8000/generate/", json=payload)
    ...

    # 2. Step through the environment to get rewards
    client.reset()
    env_rewards = []
    for msg in completions_text:
        env_result = client.step(EchoAction(message=msg))
        env_rewards.append(env_result.reward)

    # 3. Add environment rewards as extra field
    result["env_reward"] = env_rewards
    return result

def reward_from_env(completions, **kwargs):
    """Extract environment rewards passed via rollout_func kwargs."""
    env_rewards = kwargs.get("env_reward", [])
    return [float(reward) for reward in env_rewards] if env_rewards else [0.0] * len(completions)

dataset = Dataset.from_dict({"prompt": ["You are an AI that interacts with an *Echo* environment. Word to echo:"] * 64})

# Setup trainer with custom rollout
trainer = GRPOTrainer(
    model="Qwen/Qwen2.5-0.5B-Instruct",
    reward_funcs=reward_from_env,
    train_dataset=dataset,
    rollout_func=rollout_func,  # Use custom rollout
    args=GRPOConfig(
        ...
    ),
)
trainer.train()
```

Resources

- Scripts
- Notebooks
- Cookbooks
- Example guides
- Blogs
- Courses







Thanks!

Sergio Paniego Blanco (@sergiopaniego)

Machine Learning Engineer @ Hugging Face