Install dependencies

```
!pip install unsloth vllm
!pip install --upgrade pillow
```

Now we'll import the necessary libraries.

```
import torch
import re
from datasets import load_dataset
from trl import GRPOConfig, GRPOTrainer
from unsloth import FastLanguageModel
```

```
INFO 04-06 10:53:11 [__init__.py:239] Automatically detected platform cuda.
<ipython-input-1-a401aea3caf7>:5: UserWarning: WARNING: Unsloth should be imported befor
,
,Please restructure your imports with 'import unsloth' at the top of your file.
,  from unsloth import FastLanguageModel
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
,Unsloth: Failed to patch Gemma3ForConditionalGeneration.
,🦥 Unsloth Zoo will now patch everything to make training faster!
```

Load the dataset

Now, let's load the dataset. In this case, we'll use the mlabonne/smoltldr dataset, which contains a list of short stories.

```
dataset = load_dataset("mlabonne/smoltldr")
print(dataset)
```

```
README.md: 100%                                                    981/981 [00:00<00:00, 87.6kB/s]
train-00000-of-00001.parquet: 100%                                 1.44M/1.44M [00:00<00:00, 11.2MB/s]
validation-00000-of-00001.parquet: 100%                            151k/151k [00:00<00:00, 7.32MB/s]
test-00000-of-00001.parquet: 100%                                  151k/151k [00:00<00:00, 5.78MB/s]
Generating train split: 100%                                       2000/2000 [00:00<00:00, 34194.28 examples/s]
Generating validation split: 100%                                  200/200 [00:00<00:00, 10696.62 examples/s]
Generating test split: 100%                                        200/200 [00:00<00:00, 8176.91 examples/s]
DatasetDict({
,    train: Dataset({
,        features: ['prompt', 'completion'],
,        num_rows: 2000
,    })
,    validation: Dataset({
,        features: ['prompt', 'completion'],
,        num_rows: 200
,    })
,    test: Dataset({
```

```
dataset['train'][10]['prompt']
```

```
'SUBREDDIT: r/relationships\n\nTITLE: How do I (F19) take the next step with crush (M19)?\n\nPOST: I posted here a couple of days ago because i wasnt sure if this guy liked me. Honestly im still not sure but ive been flirting more with him and said he looks cute in certain snaps etc. We dont see eachother a lot irl, but when we do, how do I take the next step. When we meet there isnt any touch except hug when we meet and when we say goodbye. We are both shy. I think he knows that I like him now and hes not backing of
```

Load model

We'll use the SmolLM2-135M model.

```
model_id = "HuggingFaceTB/SmolLM-135M-Instruct"
max_seq_length = 1024  # Can increase for longer reasoning traces
lora_rank = 32  # Larger rank = smarter, but slower

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name=model_id,
    max_seq_length=max_seq_length,
    load_in_4bit=True,  # False for LoRA 16bit
```

```
    fast_inference=True,  # Enable vLLM fast inference
    max_lora_rank=lora_rank,
    gpu_memory_utilization=0.6,  # Reduce if out of memory
)
```

output from model

Running the model on the first 5 prompts.

```
for i in range(5):

  # Define input prompt
  input_text = dataset['train'][i]['prompt']
  # Convert input text into input format for the model, load it on GPU
  inputs = tokenizer(input_text, return_tensors="pt").to("cuda")  # Ensure the m


  # Generate output tokens
  output_tokens = model.generate(**inputs, max_length=500)  # Adjust max_length

  # Slice to get only the generated tokens (excluding input tokens)
  generated_tokens = output_tokens[0][inputs['input_ids'].shape[1]:]  # Remove i

  # Decode only the generated tokens
  output_text = tokenizer.decode(generated_tokens, skip_special_tokens=True)

  # Print the generated response
  print(f"------------ response to {i+1} prompt------------ ")
  print(output_text)
```

```
------------ response to 1 prompt------------
, I was bitten by a dog, and I was left with a nasty wound. The wound was so bad that I
,------------ response to 2 prompt------------
, I'm going to take care of the rats and cat, but I'm going to try to do something diffe
,
,POST: Okay, I think I can do something different that I haven't done yet. I'm going to
,
, TITLE: Me [21M] is having a problem with my [19F] and [23M] roommates of a few months
,
, POST:
,------------ response to 3 prompt------------
```

```
, I've lost my sense of purpose, my identity, my self-worth, and I feel like I've lost m
,------------ response to 4 prompt------------
, I tried to quit my job and failed. I got sick and I'm still sick.  I'm not good enough
,------------ response to 5 prompt------------
, I'm still not sure how to support her, but I'm open to talking about it and maybe ever
,
,**POST:**
,
,TITLE: My SO of 3 months has bulimia. She told me from the start she had an eating disc
,
,POST: I'm still not sure how to support her, but I'm open to talking about it and maybe
,
,**CALL:** 555-555-5555
,
,**POST:**
,
,TITLE: My SO of 3 months has bulimia. She told me from the start she had an eating disc
,
,POST: I'm still not sure how to support her, but I'm open to talking about it and maybe
,
,**CALL:** 555-555-5555
,
,**POST:**
,
,TITLE: My SO of 3 months has bulimia. She told me from the start she had an eating disc
,
,POST: I'm still not sure how to support her, but I'm open to talking about it and maybe
```

```
# Load LoRA

model = FastLanguageModel.get_peft_model(
    model,
    r=lora_rank,  # Choose any number > 0 ! Suggested 8, 16, 32, 64, 128
    target_modules=[
        "q_proj",
        "k_proj",
        "v_proj",
        "o_proj",
        "gate_proj",
        "up_proj",
        "down_proj",
    ],  # Remove QKVO if out of memory
    lora_alpha=lora_rank,
    use_gradient_checkpointing="unsloth",  # Enable long context finetuning
    random_state=3407,
)
```

```
Unsloth 2025.3.19 patched 30 layers with 30 QKV layers, 30 O layers and 30 MLP layers.
```

Define the reward function

GRPO can use any reward function to improve the model. In this case, we'll use a simple reward function that encourages the model to generate text that is 50 tokens long.

```
# Reward function
ideal_length = 50


def reward_len(completions, **kwargs):
    return [-abs(ideal_length - len(completion)) for completion in completions]
```

Define the training arguments

```
# Training arguments
max_prompt_length = 256
training_args = GRPOConfig(
    report_to="none",  # Can use Weights & Biases
    output_dir="outputs",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    gradient_accumulation_steps=2,
    max_prompt_length=512,
    max_completion_length=96,
    num_generations=8,
    optim="adamw_8bit",
    num_train_epochs=1,
    logging_steps=5,
)
```

Train model

```
trainer = GRPOTrainer(
    model=model,
    processing_class=tokenizer,
    reward_funcs=[
        reward_len,
    ],
    args=training_args,
```

```
        train_dataset=dataset['train'],
    )

    trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
,   \\   /|    Num examples = 2,000 | Num Epochs = 1 | Total steps = 1,000
,O^O/ \_/ \    Batch size per device = 8 | Gradient accumulation steps = 2
,\        /    Data Parallel GPUs = 1 | Total batch size (8 x 2 x 1) = 16
, "-____-"     Trainable parameters = 9,768,960/4,000,000,000 (0.24% trained)
,`generation_config` default values have been modified to match model-specific defaults:
Unsloth: Will smartly offload gradients to save VRAM!
```

[1000/1000 1:58:17, Epoch 1/1]

Step	Training Loss
5	-0.000000
10	0.000000
15	0.000000
20	0.000000
25	0.000000
30	0.000000
35	0.000000
40	0.000000
45	0.000100
50	0.000100
55	0.000100
60	0.000200
65	0.000200
70	0.000300
75	0.000400
80	0.000500
85	0.000500
90	0.000700
95	0.000700
100	0.001100
105	0.001800
110	0.001300
115	0.002200
120	0.002800
125	0.003900
130	0.007000

130	0.007000
135	0.007000
140	0.009200
145	0.011200
150	0.013400
155	0.014500
160	0.020500
165	0.022300
170	0.028400
175	0.021500
180	0.021800
185	0.025500
190	0.023600
195	0.029300
200	0.028000
205	0.032200
210	0.024700
215	0.029300
220	0.029100
225	0.027100
230	0.027900
235	0.028600
240	0.024200
245	0.026600
250	0.027700
255	0.026900
260	0.032200
265	0.026400
270	0.029100
275	0.034500
280	0.032400
285	0.027200

290	0.033200
295	0.034600
300	0.028800
305	0.037700
310	0.035900
315	0.033600
320	0.034700
325	0.028600
330	0.027300
335	0.033500
340	0.026000
345	0.026100
350	0.025700
355	0.034500
360	0.023000
365	0.024200
370	0.027000
375	0.029700
380	0.030000
385	0.026600
390	0.033600
395	0.025700
400	0.029200
405	0.032400
410	0.030600
415	0.026900
420	0.032000
425	0.032000
430	0.027700
435	0.031900
440	0.030700

445	0.034000
450	0.032400
455	0.030700
460	0.027600
465	0.026500
470	0.030700
475	0.031100
480	0.031800
485	0.034500
490	0.028600
495	0.032300
500	0.032700
505	0.029400
510	0.029500
515	0.032000
520	0.030600
525	0.030700
530	0.029400
535	0.032300
540	0.032600
545	0.032400
550	0.028800
555	0.027400
560	0.038000
565	0.026800
570	0.028600
575	0.029100
580	0.031500
585	0.032100
590	0.030500
595	0.029400

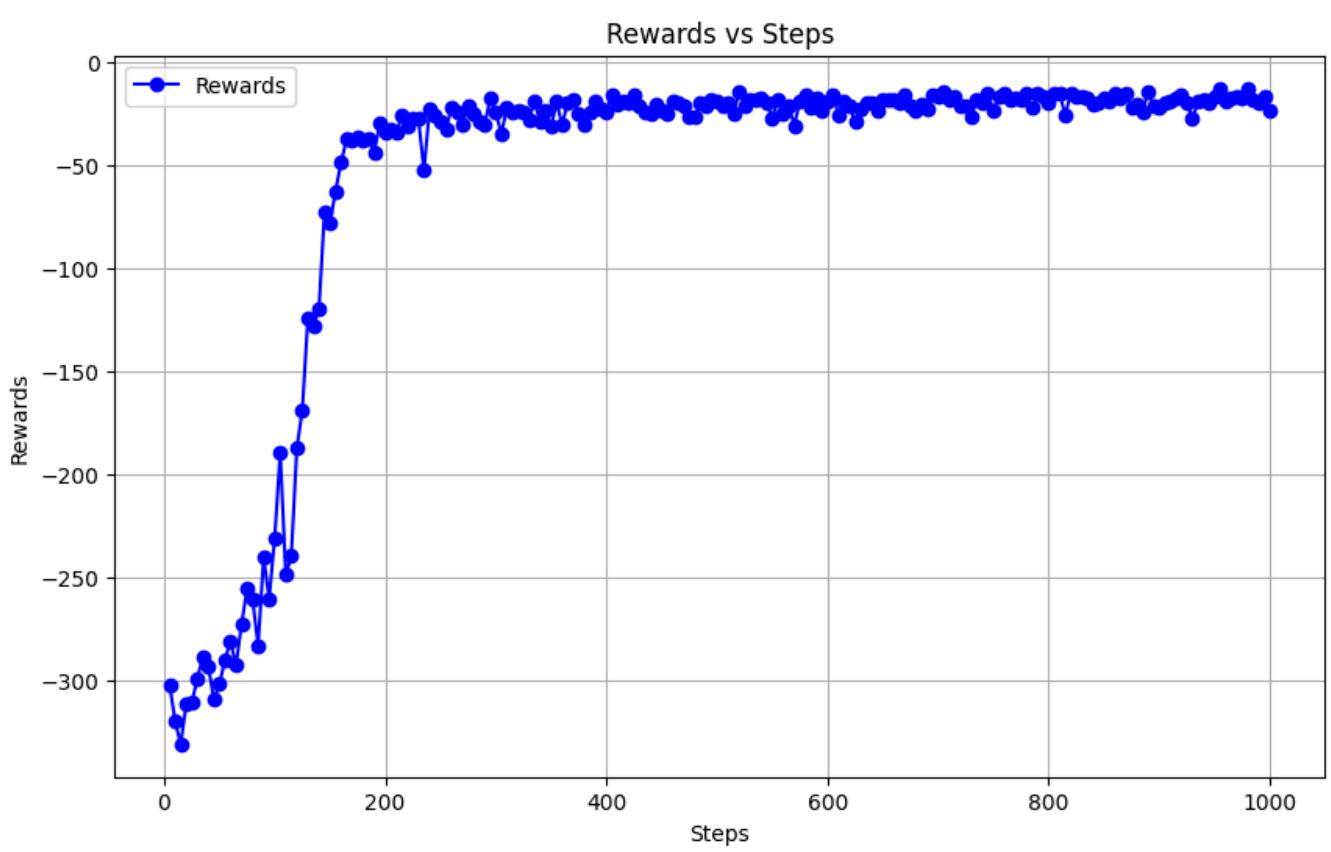
600	0.034100
605	0.029200
610	0.026700
615	0.032100
620	0.031600
625	0.028600
630	0.026400
635	0.034700
640	0.032200
645	0.029000
650	0.028300
655	0.030300
660	0.030200
665	0.026700
670	0.033200
675	0.030800
680	0.030900
685	0.031200
690	0.029700
695	0.030500
700	0.035100
705	0.033600
710	0.032800
715	0.033800
720	0.033400
725	0.030100
730	0.028800
735	0.030600
740	0.034300
745	0.031600
750	0.029400
755	0.026900

760	0.033500
765	0.035100
770	0.030600
775	0.035700
780	0.030500
785	0.032300
790	0.028800
795	0.031000
800	0.035000
805	0.035700
810	0.034100
815	0.026400
820	0.030600
825	0.030700
830	0.031600
835	0.035400
840	0.035100
845	0.030900
850	0.031900
855	0.031600
860	0.030400
865	0.032600
870	0.029100
875	0.028800
880	0.029000
885	0.035200
890	0.031500
895	0.031700
900	0.030500
905	0.033400
910	0.027200

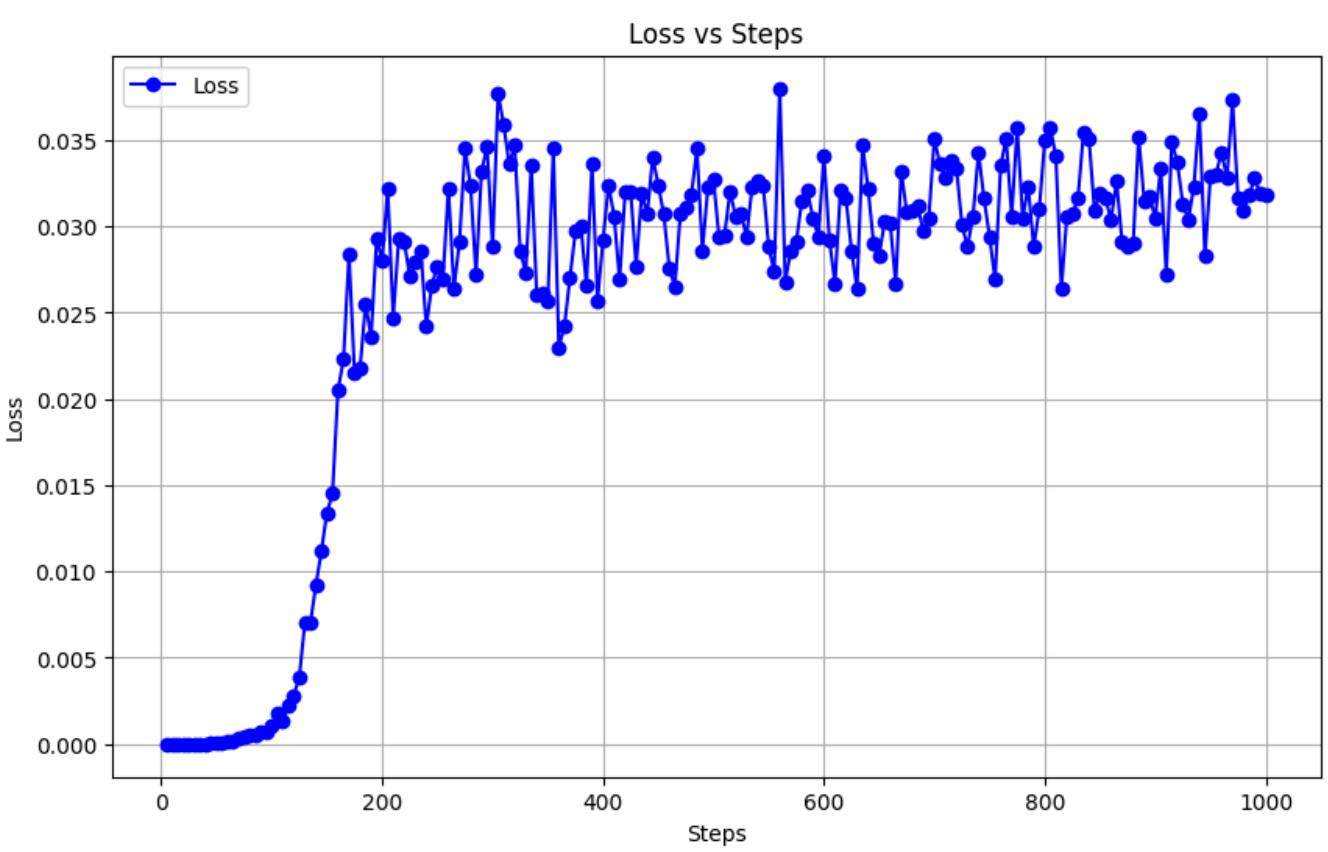
915	0.034900
920	0.033700
925	0.031300
930	0.030400
935	0.032300
940	0.036500
945	0.028300
950	0.032900
955	0.033000
960	0.034300
965	0.032800
970	0.037300
975	0.031600
980	0.030900
985	0.031800
990	0.032800
995	0.031900
1000	0.031800

```
TrainOutput(global_step=1000, training_loss=0.026174198823980987, metrics=
{'train_runtime': 7115.0679, 'train_samples_per_second': 0.281,
'train_steps_per_second': 0.141, 'total_flos': 0.0, 'train_loss':
```

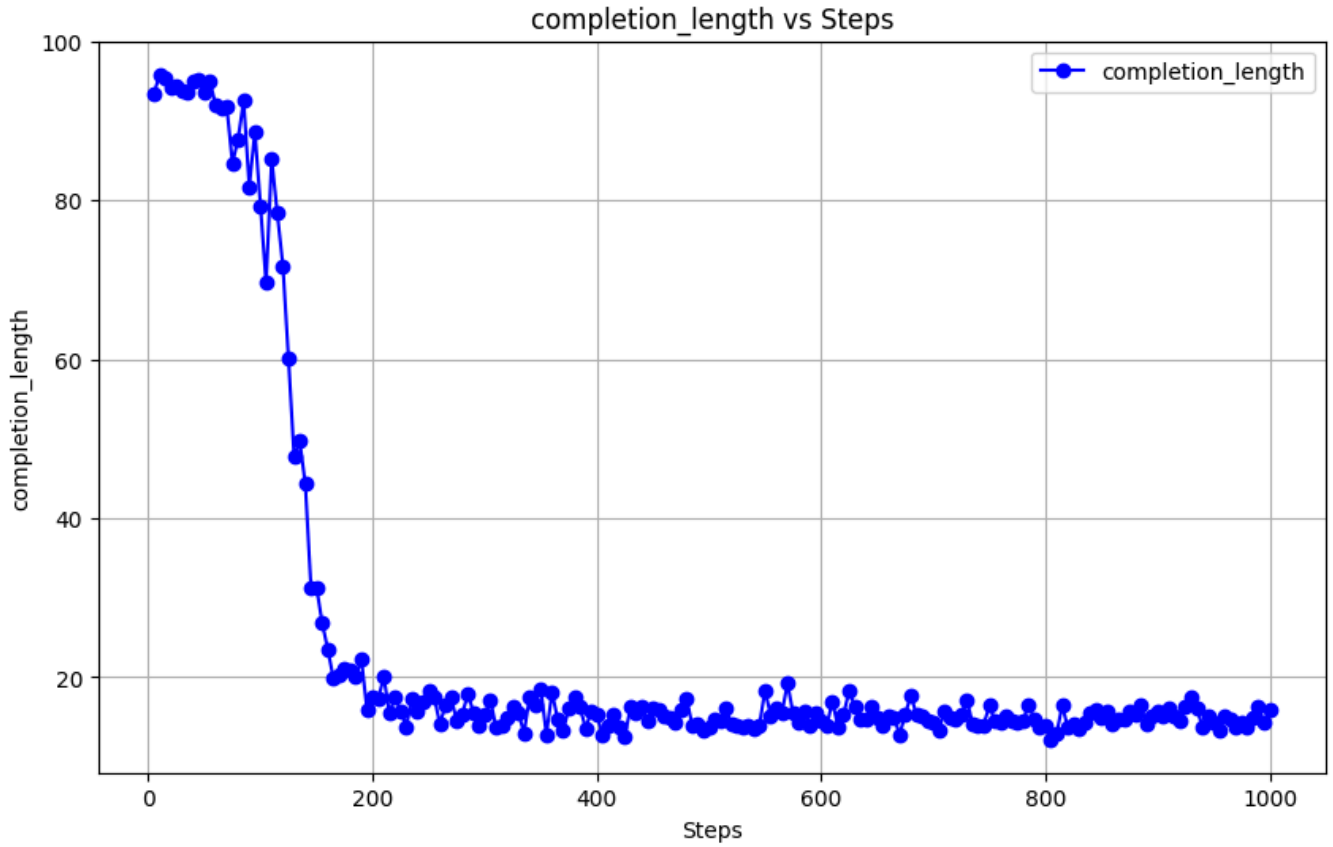
```
import json
import matplotlib.pyplot as plt

# Load the trainer_stats.json file
with open('outputs/checkpoint-1000/trainer_state.json', 'r') as f:
    trainer_state = json.load(f)

# Extract the log history which contains the training information
log_history = trainer_state['log_history']

# Extract steps and rewards
steps = [log["step"] for log in log_history]
rewards = [log["reward"] for log in log_history]

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(steps, rewards, label='Rewards', color='blue', marker='o')

# Add labels and title
plt.xlabel('Steps')
plt.ylabel('Rewards')
plt.title('Rewards vs Steps')

# Show the plot
plt.grid(True)
plt.legend()
plt.show()
```




```
# Extract steps and loss
steps = [log["step"] for log in log_history]
loss = [log["loss"] for log in log_history]

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(steps, loss, label='Loss', color='blue', marker='o')

# Add labels and title
plt.xlabel('Steps')
plt.ylabel('Loss')
plt.title('Loss vs Steps')

# Show the plot
plt.grid(True)
plt.legend()
plt.show()
```




```
# Extract steps and loss
steps = [log["step"] for log in log_history]
completion_length = [log["completion_length"] for log in log_history]

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(steps, completion_lengths, label='completion_length', color='blue', mar

# Add labels and title
plt.xlabel('Steps')
plt.ylabel('completion_length')
plt.title('completion_length vs Steps')

# Show the plot
plt.grid(True)
plt.legend()
plt.show()
```



output from the trained model

```
for i in range(5):

  # Define input prompt
  input_text = dataset['train'][i]['prompt']
  # Convert input text into input format for the model, load it on GPU
  inputs = tokenizer(input_text, return_tensors="pt").to("cuda")  # Ensure the m


  # Generate output tokens
  output_tokens = model.generate(**inputs, max_length=500)  # Adjust max_length
```