⚡ Tutorial: Train your own Reasoning model with GRPO

Beginner's Guide to transforming a model like Llama 3.1 (8B) into a reasoning model by using Unsloth and GRPO.

DeepSeek developed GRPO (Group Relative Policy Optimization) to train their R1 reasoning models.

Quickstart

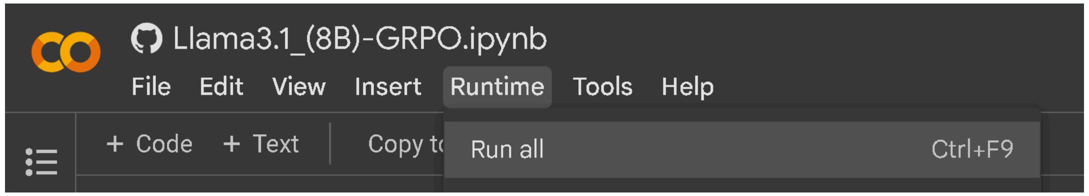
These instructions are for our pre-made Google Colab notebooks. If you are installing Unsloth locally, you can also copy our notebooks inside your favorite code editor.

The GRPO notebooks we are using: Llama 3.1 (8B), Phi-4 (14B) and Qwen2.5 (3B)

1 Install Unsloth

If you're using our Colab notebook, click **Runtime > Run all**. We'd highly recommend you checking out our Fine-tuning Guide before getting started.

If installing locally, ensure you have the correct requirements and use `pip install unsloth` on Linux or follow our Windows install instructions.



2 Learn about GRPO & Reward Functions

Before we get started, it is recommended to learn more about GRPO, reward functions and how they work. Read more about them including tips & tricks here.

You will also need enough VRAM. In general, model parameters = amount of VRAM you will need. In Colab, we are using their free 16GB VRAM GPUs which can train any model up to 16B in parameters.

3 Configure desired settings

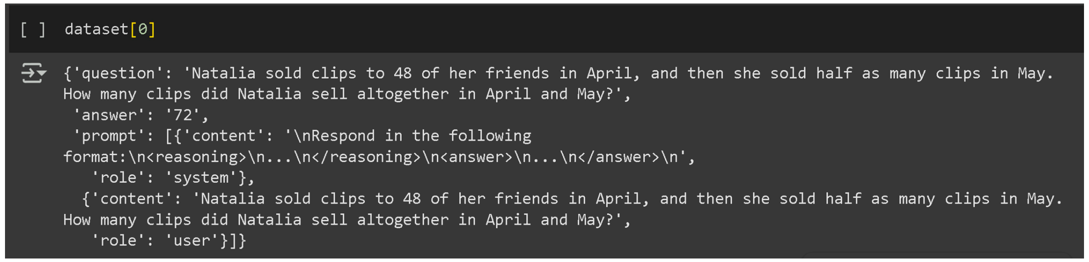
We have pre-selected optimal settings for the best results for you already and you can change the model to whichever you want listed in our supported models. Would not recommend changing other settings if you're a beginner.

```python
from unsloth import is_bfloat16_supported
import torch
max_seq_length = 512 # Can increase for longer reasoning traces
lora_rank = 32 # Larger rank = smarter, but slower

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = "meta-llama/meta-Llama-3.1-8B-Instruct",
    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
)
```

4 Select your dataset

We have pre-selected OpenAI's GSM8K dataset already but you could change it to your own or any public one on Hugging Face. You can read more about datasets here.

Your dataset should still have at least 2 columns for question and answer pairs. However the answer must not reveal the reasoning behind how it derived the answer from the question. See below for an example:



```
[ ]  dataset[0]

{'question': 'Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May.
How many clips did Natalia sell altogether in April and May?',
 'answer': '72',
 'prompt': [{'content': '\nRespond in the following
format:\n<reasoning>\n...\n</reasoning>\n<answer>\n...\n</answer>\n',
   'role': 'system'},
  {'content': 'Natalia sold clips to 48 of her friends in April, and then she sold half as many clips in May.
How many clips did Natalia sell altogether in April and May?',
   'role': 'user'}]}
```

5 Reward Functions/Verifier

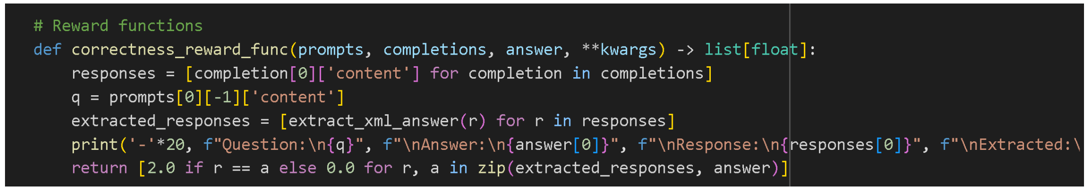
Reward Functions/Verifiers lets us know if the model is doing well or not according to the dataset you have provided. Each generation run will be assessed on how it performs to the score of the average of the rest of generations. You can create your own reward functions however we have already pre-selected them for you with Will's GSM8K reward functions. With this, we have 5 different ways which we can reward each generation.

You can input your generations into an LLM like ChatGPT 4o or Llama 3.1 (8B) and design a reward function and verifier to evaluate it. For example, feed your generations into a LLM of your choice and set a rule: "If the answer sounds too robotic, deduct 3 points." This helps refine outputs based on quality criteria. **See examples** of what they can look like here.

Example Reward Function for an Email Automation Task:

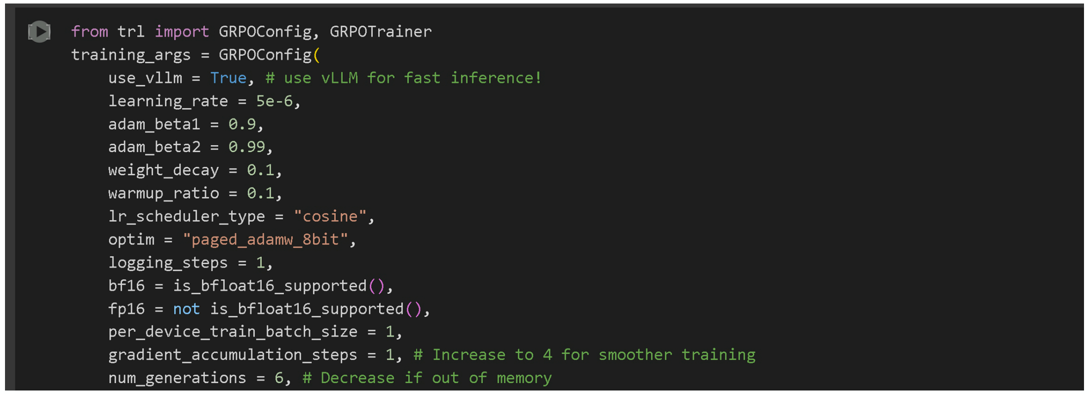
- **Question:** Inbound email
- **Answer:** Outbound email
- **Reward Functions:**
  - If the answer contains a required keyword → **+1**
  - If the answer exactly matches the ideal response → **+1**
  - If the response is too long → **-1**
  - If the recipient's name is included → **+1**

  - If a signature block (phone, email, address) is present → **+1**


```python
# Reward functions
def correctness_reward_func(prompts, completions, answer, **kwargs) -> list[float]:
    responses = [completion[0]['content'] for completion in completions]
    q = prompts[0][-1]['content']
    extracted_responses = [extract_xml_answer(r) for r in responses]
    print('-'*20, f"Question:\n{q}", f"\nAnswer:\n{answer[0]}", f"\nResponse:\n{responses[0]}", f"\nExtracted:\
    return [2.0 if r == a else 0.0 for r, a in zip(extracted_responses, answer)]
```

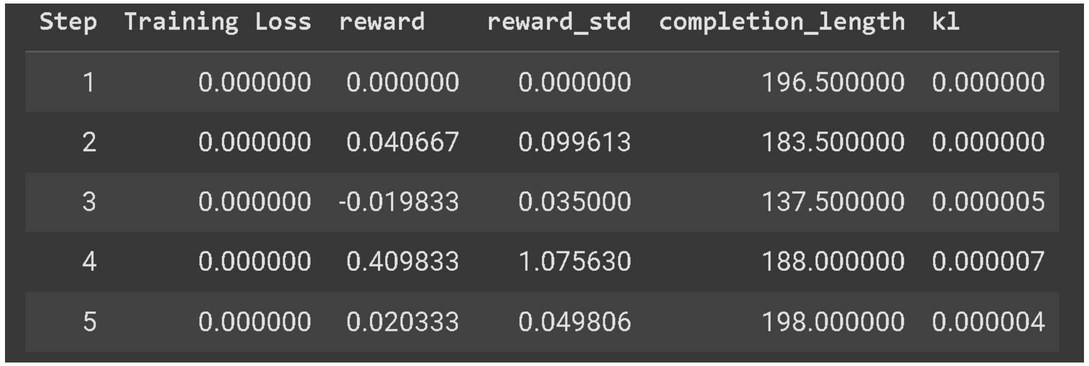
6 Train your model

We have pre-selected hyperparameters for the most optimal results however you could change them. Read all about parameters here.


```python
from trl import GRPOConfig, GRPOTrainer
training_args = GRPOConfig(
    use_vllm = True, # use vLLM for fast inference!
    learning_rate = 5e-6,
    adam_beta1 = 0.9,
    adam_beta2 = 0.99,
    weight_decay = 0.1,
    warmup_ratio = 0.1,
    lr_scheduler_type = "cosine",
    optim = "paged_adamw_8bit",
    logging_steps = 1,
    bf16 = is_bfloat16_supported(),
    fp16 = not is_bfloat16_supported(),
    per_device_train_batch_size = 1,
    gradient_accumulation_steps = 1, # Increase to 4 for smoother training
    num_generations = 6, # Decrease if out of memory
```

You should see the reward increase overtime. We would recommend you train for at least 300 steps which may take 30 mins however, for optimal results, you should train for longer.

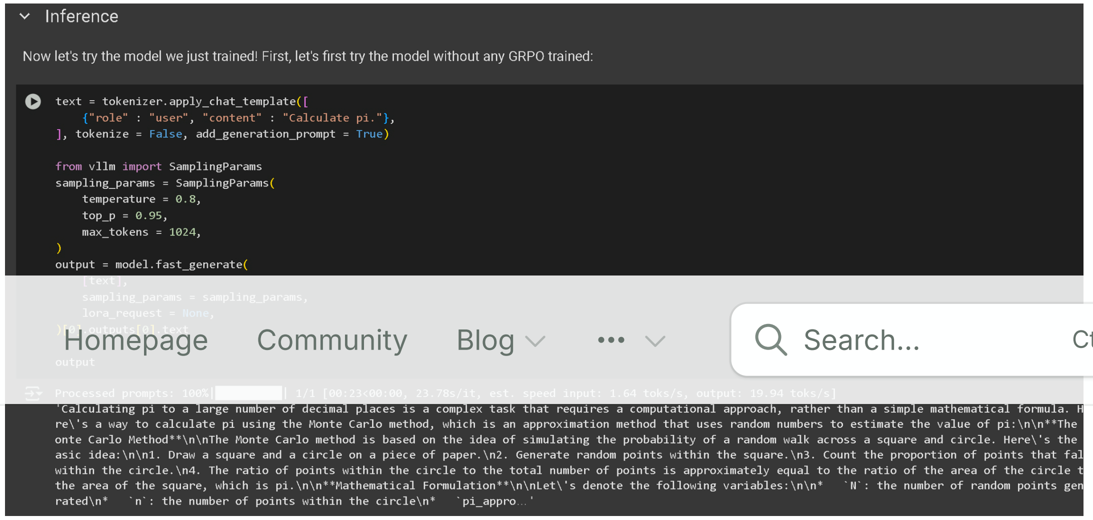
You will also see sample answers which allows you to see how the model is learning. Some may have steps, XML tags, attempts etc. and the idea is as trains it's going to get better and better because it's going to get scored higher and higher until we get the outputs we desire with long reasoning chains of answers.



Step	Training Loss	reward	reward_std	completion_length	kl
1	0.000000	0.000000	0.000000	196.500000	0.000000
2	0.000000	0.040667	0.099613	183.500000	0.000000
3	0.000000	-0.019833	0.035000	137.500000	0.000005
4	0.000000	0.409833	1.075630	188.000000	0.000007
5	0.000000	0.020333	0.049806	198.000000	0.000004

7 Run & Save your model

Run your model by clicking the play button. In the first example, there is usually no reasoning in the answer and in order to see the reasoning, we need to first save the LoRA we just trained with GRPO first.



Inference

Now let's try the model we just trained! First, let's first try the model without any GRPO trained:

```
text = tokenizer.apply_chat_template([
    {"role" : "user", "content" : "Calculate pi."},
], tokenize = False, add_generation_prompt = True)

from vllm import SamplingParams
sampling_params = SamplingParams(
    temperature = 0.8,
    top_p = 0.95,
    max_tokens = 1024,
)
output = model.fast_generate(
    [text],
    sampling_params = sampling_params,
    lora_request = None,
)[0].outputs[0].text

output
```

```
Processed prompts: 100%|██████████| 1/1 [00:23<00:00, 23.78s/it, est. speed input: 1.64 toks/s, output: 19.94 toks/s]
'Calculating pi to a large number of decimal places is a complex task that requires a computational approach, rather than a simple mathematical formula. H
re\'s a way to calculate pi using the Monte Carlo method, which is an approximation method that uses random numbers to estimate the value of pi:\n\n**The
onte Carlo Method**\n\nThe Monte Carlo method is based on the idea of simulating the probability of a random walk across a square and circle. Here\'s the
asic idea:\n\n1. Draw a square and a circle on a piece of paper.\n2. Generate random points within the square.\n3. Count the proportion of points that fal
within the circle.\n4. The ratio of points within the circle to the total number of points is approximately equal to the ratio of the area of the circle t
the area of the square, which is pi.\n\n**Mathematical Formulation**\n\nLet\'s denote the following variables:\n\n*   `N`: the number of random points gen
rated\n*   `n`: the number of points within the circle\n*   `pi_appro...'
```

Unsloth Documentation

Homepage Community Blog ••• Search... Ctrl + K

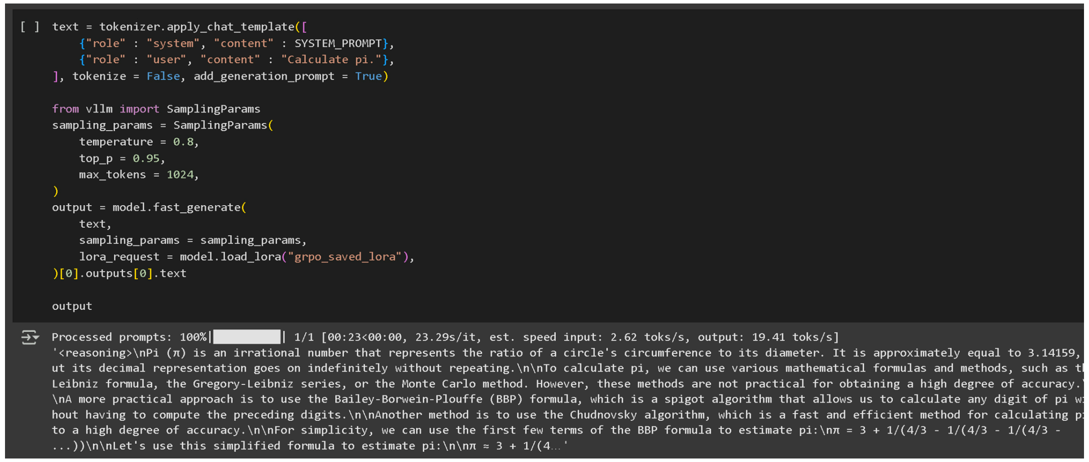
GET STARTED

👋 Welcome

The first inference example run has no reasoning. You must load the LoRA and test it to reveal the reasoning.

Then we load the LoRA and test it. Our reasoning model is much better - it's not always correct, since we only trained it for an hour or so - it'll be better if we extend the sequence length and train for longer!

You can then save your model to GGUF, Ollama etc. by following our guide here.



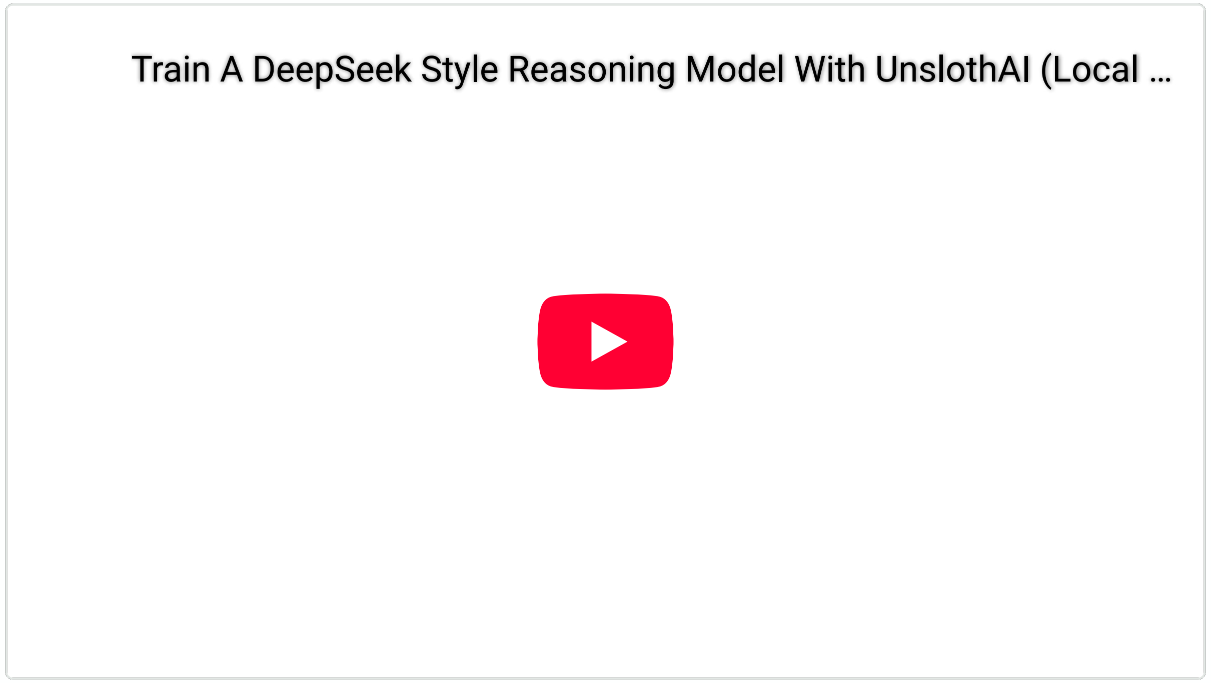
```
[ ] text = tokenizer.apply_chat_template([
        {"role" : "system", "content" : SYSTEM_PROMPT},
        {"role" : "user", "content" : "Calculate pi."},
    ], tokenize = False, add_generation_prompt = True)

    from vllm import SamplingParams
    sampling_params = SamplingParams(
        temperature = 0.8,
        top_p = 0.95,
        max_tokens = 1024,
    )
    output = model.fast_generate(
        text,
        sampling_params = sampling_params,
        lora_request = model.load_lora("grpo_saved_lora"),
    )[0].outputs[0].text

    output
```

```
Processed prompts: 100%|██████████| 1/1 [00:23<00:00, 23.29s/it, est. speed input: 2.62 toks/s, output: 19.41 toks/s]
'<reasoning>\nPi (π) is an irrational number that represents the ratio of a circle's circumference to its diameter. It is approximately equal to 3.14159,
ut its decimal representation goes on indefinitely without repeating.\n\nTo calculate pi, we can use various mathematical formulas and methods, such as t
Leibniz formula, the Gregory-Leibniz series, or the Monte Carlo method. However, these methods are not practical for obtaining a high degree of accuracy.
\nA more practical approach is to use the Bailey-Borwein-Plouffe (BBP) formula, which is a spigot algorithm that allows us to calculate any digit of pi w
hout having to compute the preceding digits.\n\nAnother method is to use the Chudnovsky algorithm, which is a fast and efficient method for calculating p
to a high degree of accuracy.\n\nFor simplicity, we can use the first few terms of the BBP formula to estimate pi:\nπ = 3 + 1/(4/3 - 1/(4/3 - 1/(4/3 -
...))\n\nLet's use this simplified formula to estimate pi:\n\nπ ≈ 3 + 1/(4...'
```

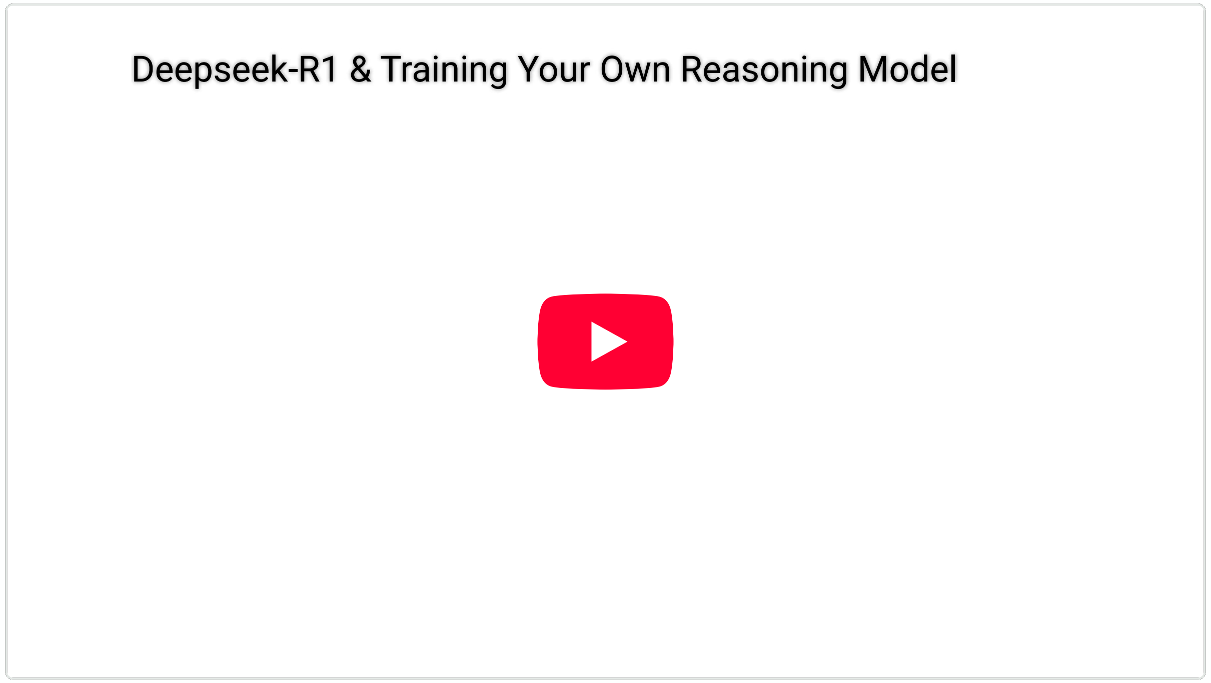
If you are still not getting any reasoning, you may have either trained for too less steps or your reward function/verifier was not optimal.

Video Tutorials

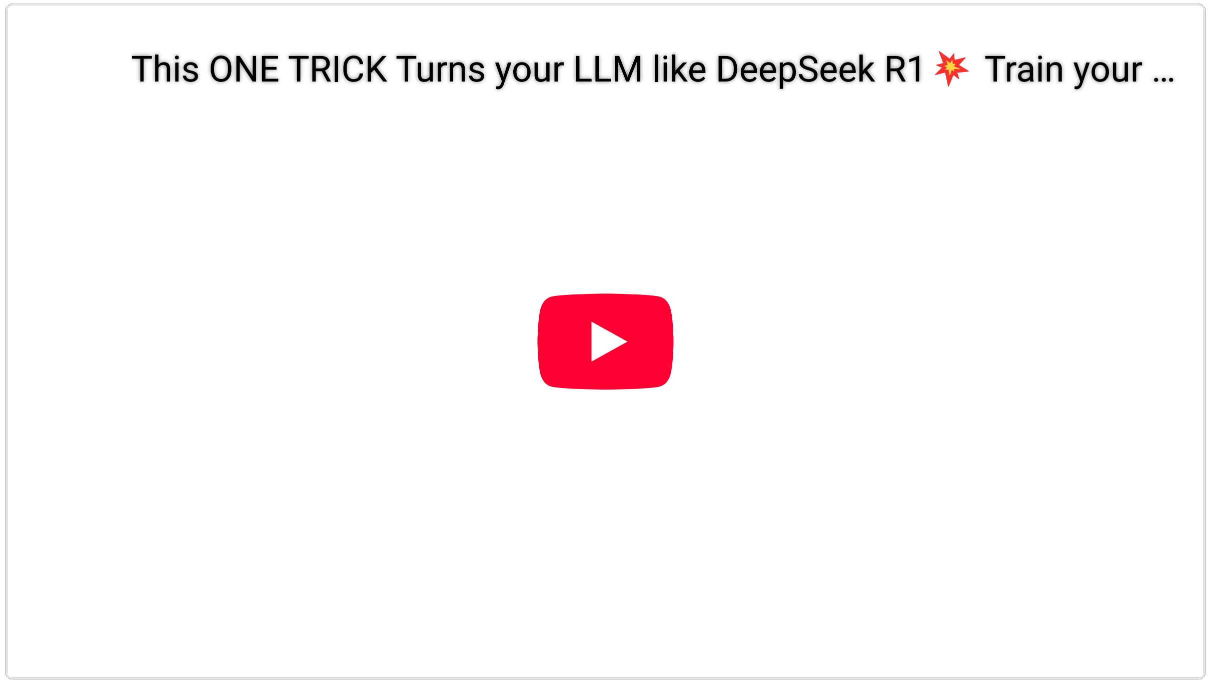
Here are some video tutorials created by amazing YouTubers who we think are fantastic!

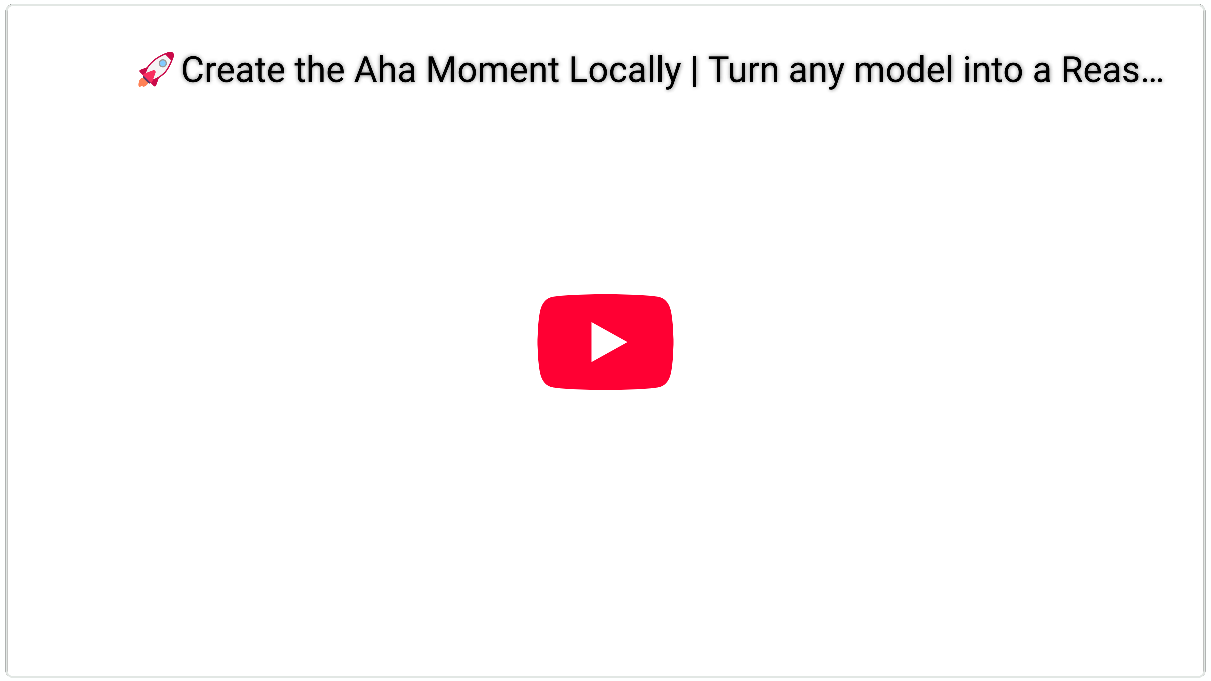

Train A DeepSeek Style Reasoning Model With UnslothAI (Local ...

Local GRPO on your own device



Great to learn about how to prep your dataset and explanations behind Reinforcement Learning + GRPO basics


This ONE TRICK Turns your LLM like DeepSeek R1 💥 Train your ...


🚀 Create the Aha Moment Locally | Turn any model into a Reas...

Previous
Reasoning - GRPO & RL

Next
Reinforcement Learning - DPO, ORPO & KTO