



Blog

Train your own R1 reasoning model with Unsloth (GRPO)

Feb 6, 2025 • By Daniel & Michael

Today, we're excited to introduce reasoning in [Unsloth](#)! DeepSeek's R1 research revealed an "aha moment" where R1-Zero autonomously learned to allocate more thinking time without human feedback by using Group Relative Policy Optimization (GRPO).



We've enhanced the entire GRPO process, making it use 80% less VRAM than Hugging Face + FA2. This allows you to reproduce R1-Zero's "aha moment" on just 7GB of VRAM using Qwen2.5 (1.5B).

Try our free GRPO notebook:

[Llama 3.1 \(8B\) on Colab](#)

For GRPO notebooks featuring other models like Phi-4, please visit our [docs](#)

[Join Our Discord](#)



❤️ P.S. thanks for the love on our R1
Dynamic [1.58-bit](#) GGUF last week & don't
forget to ★ Star us:
github.com/unslothai/unsloth



Main Details

1. With 15GB VRAM, Unsloth allows you to transform any model up to 15B parameters like Llama 3.1 (8B), Phi-4 (14B), Mistral (7B) or Qwen2.5 (7B) into a reasoning model
2. Minimum requirement: Just 7GB VRAM is enough to train your own reasoning model locally.
3. The incredible team at [Tiny-Zero](#) demonstrated that you could achieve your own "aha" moment with Qwen2.5 (1.5B) - but it required 2xA100 GPUs (160GB VRAM). Now, with Unsloth, you can achieve the same "aha" moment using just a single 7GB VRAM GPU
4. Previously, GRPO was only supported for full fine-tuning, but we've made it work with QLoRA and LoRA
5. Please note, this isn't fine-tuning DeepSeek's R1 distilled models or using distilled data from R1 for tuning which Unsloth already supported. This is converting a standard model into a full-fledged reasoning model using GRPO.
6. Use cases for GRPO include: If you want to make a customized model with rewards (say for law, medicine etc.), then GRPO can help.



[Join Our Discord](#)



If you have input and output data (like questions and answers), but do not have the chain of thought or reasoning process, GRPO can magically create the reasoning process for you! + much more

GRPO + The "aha" moment

DeepSeek's researchers observed an "aha moment" when training R1-Zero with pure reinforcement learning (RL). The model learned to extend its thinking time by reevaluating its initial approach, without any human guidance or predefined instructions.

In a test example, even though we only trained Phi-4 with 100 steps using GRPO, the results are already clear. The model without GRPO does not have the thinking token, whilst the one trained with GRPO does and also has the correct answer.



Prompt: Which is bigger? 9.11 or 9.9?

Before (Phi-4 with no reasoning)

9.11 is bigger than 9.9. When comparing decimal numbers, you look at the digits from left to right. Both numbers have the same whole number part (9), so you compare the digits in the tenths place next. In 9.11, the tenths place is 1, and in 9.9, the tenths place is 9. Since 1 is less than 9, you might initially think 9.9 is larger, but you also need to consider the hundredths place in 9.11, which is 1. When you express 9.9 as 9.90 for comparison, you see that 9.11 is greater than 9.90. Therefore, 9.11 is bigger than 9.9.

After (Phi-4 with GRPO)

<reasoning>
To determine which number is bigger between 9.11 and 9.9, we should compare the two numbers digit by digit from left to right.

1. First, compare the digits in the units place:
 - Both numbers have a 9 in the units place.
2. Next, compare the digits in the tenths place:
 - The number 9.11 has a 1 in the tenths place.
 - The number 9.9 has a 9 in the tenths place.
- Since 1 is less than 9, the number 9.11 is less than 9.9 based on the tenths place comparison.
3. For thoroughness, consider the hundredths place:
 - The number 9.11 has a 1 in the hundredths place.
 - The number 9.9 can be written as 9.90, which has a 0 in the hundredths place.

Even if we compare the hundredths place, 1 is greater than 0, but this is irrelevant since the comparison in the tenths place already determines that 9.11 is smaller than 9.9. Therefore, 9.9 is greater than 9.11.

</reasoning>

<answer>
9.9 is bigger than 9.11.
</answer>

Join Our Discord



This magic could be recreated through GRPO, a RL algorithm that optimizes responses efficiently without requiring a value function, unlike Proximal Policy Optimization (PPO) which relies on a value function. In our notebooks, we train a model with GRPO, aiming for it to develop its own self-verification and search abilities autonomously - creating a mini "aha moment".

How it works:

1. The model generates groups of responses.
2. Each response is scored based on correctness or another metric created by some set reward function rather than an LLM reward model.
3. The average score of the group is computed.
4. Each response's score is compared to the group average.
5. The model is reinforced to favor higher-scoring responses.



As an example, assume we want a model to solve:

What is 1+1? >> Chain of thought/working out >> The answer is 2.

What is 2+2? >> Chain of thought/working out >> The answer is 4.

Originally, one had to collect large swathes of data to fill the working out / chain of

Join Our Discord



thought process. But GRPO (the algorithm DeepSeek uses) or other RL algorithms can steer the model to automatically exhibit reasoning capabilities and create the reasoning trace. Instead, we need to create good reward functions or verifiers. For example, if it gets the correct answer, give it a score of 1. If some words are mis-spelt, minus 0.1. And so on! We can provide many many functions to reward the process.



GRPO in Unsloth

If you're using GRPO with Unsloth locally, please "pip install diffusers" as well as it is a dependency.

Wait for at least 300 steps for the reward to actually increase and please use the latest version of vLLM. Keep in mind that our example on Colab was just trained in an hour so the results are subpar. In order to get good results, you will need to train for at least 12 hours (this is how GRPO works), but keep in mind this isn't compulsory as you can stop at anytime.

It's advised to apply GRPO to a model at least 1.5B in parameters to correctly generate thinking tokens as smaller models may not. If you're using a base model, ensure you have a chat template. Training loss tracking for GRPO is now built directly into Unsloth, eliminating the need for external tools like wandb etc.

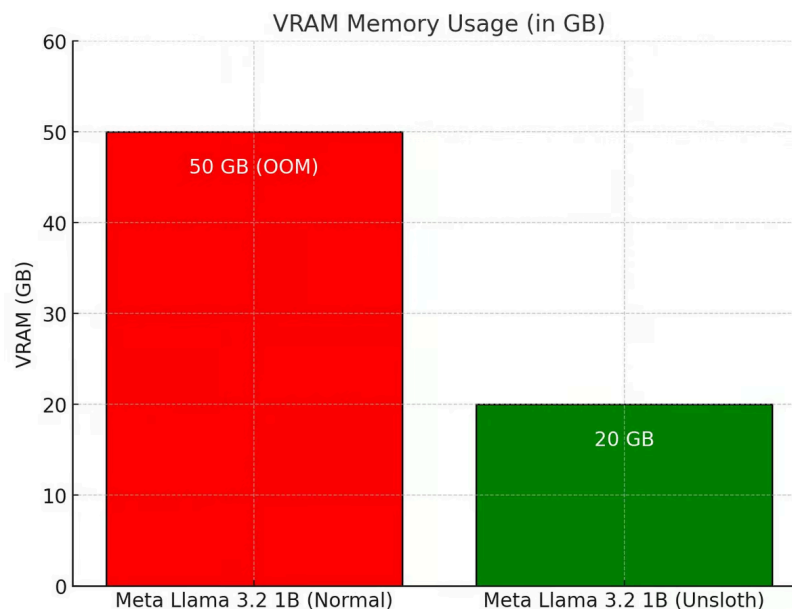




Progress: [19/100 27:23 < 2:10:29, 0.01 it/s, Epoch 0.00/1]

Step	Training Loss	reward	reward_std	completion_length	k1
1	0.000000	0.041667	0.064550	200.000000	0.000000
2	0.000000	0.125000	0.000000	200.000000	0.000000
3	0.000000	0.404500	0.960518	191.666672	0.000002
4	0.000000	0.104167	0.051031	200.000000	0.000002
5	0.000000	0.104167	0.051031	200.000000	0.000002
6	0.000000	0.044500	0.197184	200.000000	0.000002
7	0.000000	0.044167	0.145617	198.500000	0.000004

In addition to adding GRPO support, we subsequently have support for Online DPO, PPO and RLOO as well! More details can be seen in [Keith's post](#) and [blog](#) that includes the Github fork on how he got Online DPO working. The initial draft of GRPO changes on Google Colab can be seen in [Joey's tweet](#) as well! Both of their contributions allowed us to also make support for the other generation based RL methods. See below for a graph comparison of Unsloth's Online DPO VRAM consumption vs standard Hugging Face + FA2.



 **Unsloth x vLLM**

[Join Our Discord](#)



20x more throughput, 50% VRAM savings:

You can now use [vLLM](#) directly in your finetuning stack, which allows for much more throughput and allows you to finetune and do inference on the model at the same time! On 1x A100 40GB, expect **4000 tokens / s** or so with Unsloth's dynamic 4bit quant of Llama 3.2 3B Instruct. On a 16GB Tesla T4 (free Colab GPU), you can get 300 tokens / s.

We also magically **removed double memory usage** when loading vLLM and Unsloth together, allowing for savings of 5GB or so for Llama 3.1 8B and 3GB for Llama 3.2 3B (thanks to inspiration from [Boris](#)). Unsloth could originally finetune Llama 3.3 70B Instruct in 1x 48GB GPU with Llama 3.3 70B weights taking 40GB of VRAM. If we do not remove double memory usage, then we'll need ≥ 80 GB of VRAM when loading Unsloth and vLLM together.



But with Unsloth, you can still finetune and get the benefits of fast inference in one package in under 48GB of VRAM! To use fast inference, first install vllm, and instantiate Unsloth with fast_inference:

```
pip install unsloth vllm
from unsloth import FastLanguageModel
model, tokenizer =
FastLanguageModel.from_pretrained(
    model_name = "unsloth/Llama-3.2-3B-
Instruct",
    fast_inference = True,
)
model.fast_generate(["Hello!"])
```

[Join Our Discord](#)



vLLM Findings in Unsloth

1. vLLM can now load Unsloth Dynamic 4-bit quants. Just like our 1.58bit Dynamic R1 GGUF, we showed that dynamically quantizing certain layers to 4-bit and some to 16-bit can dramatically improve accuracy whilst keeping the model small.
2. We auto select multiple parameters to account for RAM, VRAM efficiency and maximum throughput (like # of chunked prefill tokens, # max sequences etc). We enable -O3 in vLLM by default and enable prefix caching. We found Flashinfer on old GPUs to be actually 10% slower. FP8 KV cache makes things 10% slower, but doubles throughput potential.
3. We allow LoRA loading in vLLM via parsing a state dict instead of loading from disk - this can make your GRPO training runs **1.5x faster**. An active area of research is to somehow directly edit the LoRA adapters in vLLM (I'm not sure how to yet). This can boost speeds a lot, since we're doing unnecessary GPU data movement now.
4. vLLM will have random VRAM spikes weirdly, especially during batched generation. We added a batched generate function to reduce memory spikes.



 **Thank you!**

[Join Our Discord](#)



A huge thank you to [Keith](#), [Edd](#), [Datta](#), [MrDragonFox](#) and [Joey](#) for their amazing help with this project. And, of course, the incredible folks at Hugging Face especially the TRL team, vLLM, and the open-source community for their contributions to making this possible. As usual, a huge thank you to everyone for using & sharing Unsloth. 🙏

As always, be sure to join our [Reddit page](#) and [Discord](#) server for help or just to show your support! You can also follow us on [Twitter](#) and [newsletter](#).

Thank you for reading!

Daniel & Michael Han 🐉

6 Feb 2025



Fine-tune Vision models now!

Get started for free

[Join Our Discord](#)