Build your own reasoning model (like deepseek -R1) 100% locally: 👉

```
query = "which is bigger 9.11 or 9.9?"

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


output = model.fast_generate(text,
                             lora_request = model.load_lora("grpo_saved_lora"),
                             )[0].outputs[0].text

print(output)
```

```
<comparing two decimal numbers>
Since both numbers have the same integer part (9), we need to compare their fractional parts (0.11 and 0.9).
0.9 is greater than 0.11 because 9 is greater than 11.
</comparing two decimal numbers>
<answer>
9.9
</answer>
```

Llama 3 produces a response with reasoning

Preview

Regular Llama 3 produces incorrect response with no reasoning

```
query = "which is bigger 9.11 or 9.9?"

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


output = model.fast_generate(text,
                             lora_request = None
                             )[0].outputs[0].text

print(output)
```

```
9.11 is bigger than 9.9.
```

Preview

Llama 3 (trained with reasoning) produces correct output with reasoning steps

```
query = "which is bigger 9.11 or 9.9?"

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


output = model.fast_generate(text,
                             lora_request = model.load_lora("grpo_saved_lora"),
                             )[0].outputs[0].text

print(output)
```

```
<comparing two decimal numbers>
Since both numbers have the same integer part (9), we need to compare their fractional parts (0.11 and 0.9).
0.9 is greater than 0.11 because 9 is greater than 11.
</comparing two decimal numbers>
<answer>
9.9
</answer>
```

Implementation ahead 👉

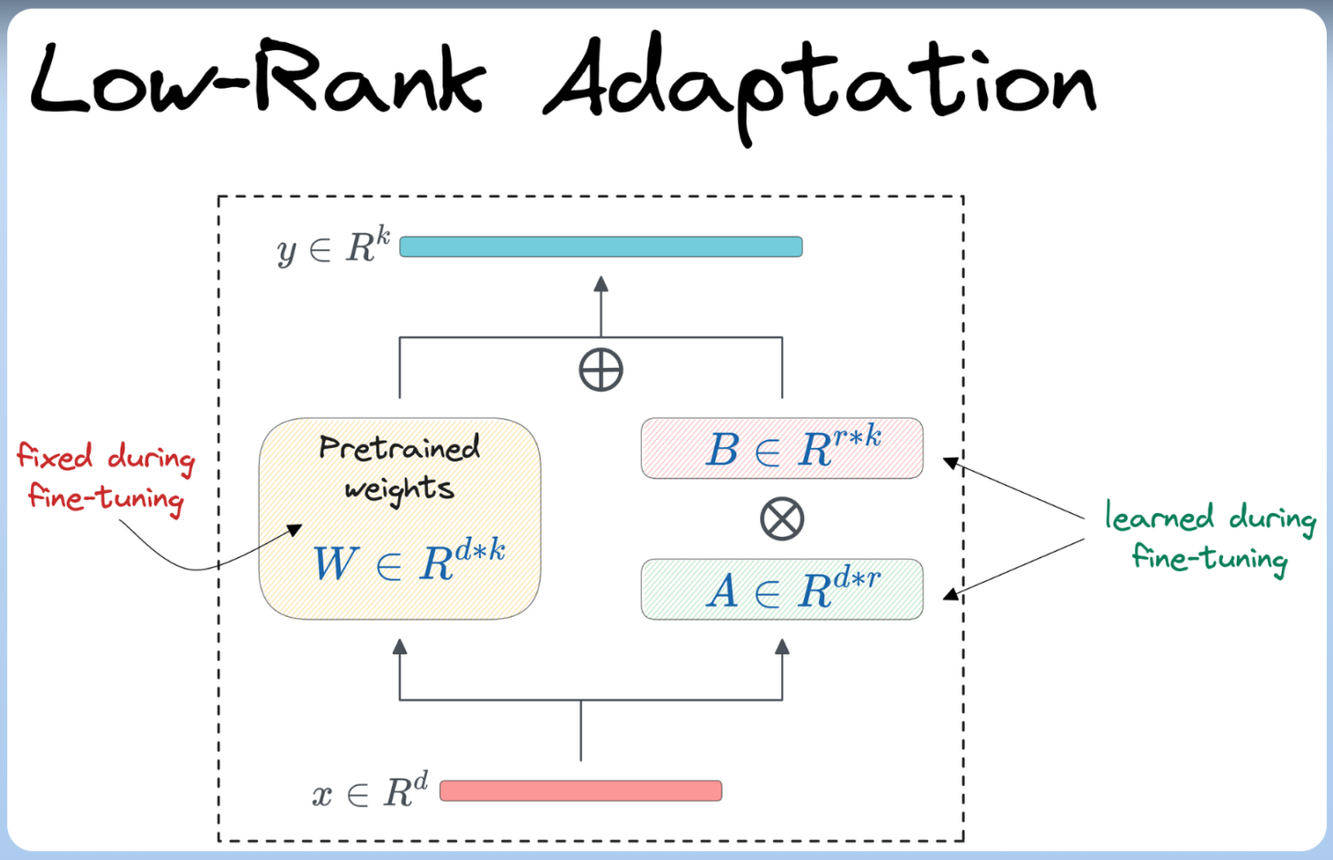
1) Load LLM

LLaMA by Meta

```
# pip install unsloth vllm

from unsloth import FastLanguageModel
import torch

MODEL = "meta-llama/meta-Llama-3.1-8B-Instruct"

model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = MODEL,
    max_seq_length = 512,
    load_in_4bit = True,
    fast_inference = True,
    max_lora_rank = 32,
)
```

2) Define LoRA config

```
model = FastLanguageModel.get_peft_model(
    model,
    r = 16,
    target_modules = ["q_proj", "k_proj", "v_proj", "o_proj"],
    use_gradient_checkpointing = "unsloth",
    lora_alpha = 16,
)
```



```python
from datasets import load_dataset

# Load and prep dataset
SYSTEM_PROMPT = """
Respond in the following format:
<reasoning>
...
</reasoning>
<answer>
...
</answer>
"""
```

3) Prepare dataset

```python
def create_dataset(split = "train"):
    data = load_dataset('openai/gsm8k', 'main')[split]
    data = data.map(lambda x: {
        'prompt': [
            {'role': 'system', 'content': SYSTEM_PROMPT},
            {'role': 'user', 'content': x['question']}
        ],
        'answer': extract_hash_answer(x['answer'])
    })
    return data

dataset = create_dataset()
```

Data sample

```
>>> dataset[1]

{'answer': '5',
 'prompt': [{'content': '\n'
                        'Respond in the following format:\n'
                        '<reasoning>\n'
                        '...\n'
                        '</reasoning>\n'
                        '<answer>\n'
                        '...\n'
                        '</answer>\n',
             'role': 'system'},
            {'content': 'Joy can read 8 pages of a book in 20 minutes. How '
                        'many hours will it take her to read 120 pages?',
             'role': 'user'}],
 'question': 'Joy can read 8 pages of a book in 20 minutes. How many hours '
             'will it take her to read 120 pages?'}
```

4) Define Reward functions

```python
def correctness_reward_func(prompts, completions, answer):
    responses = [completion[0]['content'] for completion in completions]
    q = prompts[0][-1]['content']
    extracted_responses = [extract_xml_answer(r) for r in responses]
    return [2.0 if r == a else 0.0 for r, a in zip(extracted_responses, answer)]

def int_reward_func(completions):
    responses = [completion[0]['content'] for completion in completions]
    extracted_responses = [extract_xml_answer(r) for r in responses]
    return [0.5 if r.isdigit() else 0.0 for r in extracted_responses]

def strict_format_reward_func(completions):
    pattern = r"^<reasoning>\n.*?\n</reasoning>\n<answer>\n.*?\n</answer>\n$"
    responses = [completion[0]["content"] for completion in completions]
    matches = [re.match(pattern, r) for r in responses]
    return [0.5 if match else 0.0 for match in matches]

def soft_format_reward_func(completions):
    pattern = r"<reasoning>.*?</reasoning>\s*<answer>.*?</answer>"
    responses = [completion[0]["content"] for completion in completions]
    matches = [re.match(pattern, r) for r in responses]
    return [0.5 if match else 0.0 for match in matches]

def count_xml(text):
    count = 0.0
    if text.count("<reasoning>\n") == 1:
        count += 0.125
    if text.count("\n</reasoning>\n") == 1:
        count += 0.125
    if text.count("\n<answer>\n") == 1:
        count += 0.125
        count -= len(text.split("\n</answer>\n")[-1])*0.001
    if text.count("\n</answer>") == 1:
        count += 0.125
        count -= (len(text.split("\n</answer>")[-1]) - 1)*0.001
    return count

def xmlcount_reward_func(completions):
    contents = [completion[0]["content"] for completion in completions]
    return [count_xml(c) for c in contents]
```

inspired from @willccbb

5) Define GRPO config and trainer

```python
from trl import GRPOConfig

training_args = GRPOConfig(
    use_vllm = True,
    learning_rate = 5e-6,
    weight_decay = 0.1,
    warmup_ratio = 0.1,
    lr_scheduler_type = "cosine",
    optim = "paged_adamw_8bit",
    gradient_accumulation_steps = 1,
    num_generations = 6,
    max_completion_length = 200,
    max_steps = 250,
)
```

```python
from trl import GRPOTrainer

trainer = GRPOTrainer(
    model = model,
    processing_class = tokenizer,
    reward_funcs = [
        xmlcount_reward_func,
        soft_format_reward_func,
        strict_format_reward_func,
        int_reward_func,
        correctness_reward_func,
    ],
    args = training_args,
    train_dataset = dataset,
)
```

6) Train

```
trainer.train()
```

```
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs = 1
   \\   /|    Num examples = 7,473 | Num Epochs = 1
O^O/ \_/ \    Batch size per device = 1 | Gradient Accumulation steps = 1
\        /    Total batch size = 1 | Total steps = 250
 "-____-"     Number of trainable parameters = 27,262,976
```

[91/250 1:11:39 < 2:08:00, 0.02 it/s, Epoch 0.01/1]

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	137.000000	0.000005
4	0.000000	0.166667	0.258199	189.166672	0.000006
5	0.000000	0.000000	0.000000	192.500000	0.000004
6	0.000000	0.029333	0.050274	144.833344	0.000005
7	0.000000	0.367833	1.051099	159.333344	0.000007
8	0.000000	0.777667	1.157384	171.833344	0.000007
9	0.000000	-0.034000	0.083283	149.500000	0.000008
10	0.000000	0.447167	1.008919	94.333336	0.000010

Comparison

```
query = "which is bigger 9.11 or 9.9?"

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


output = model.fast_generate(text,
                             lora_request = None
                )[0].outputs[0].text

print(output)
```

incorrect answer

```
9.11 is bigger than 9.9.
```

Regular Llama 3 produces incorrect response with no reasoning

Comparison

```
query = "which is bigger 9.11 or 9.9?"

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


output = model.fast_generate(text,
                             lora_request = model.load_lora("grpo_saved_lora"),
                             )[0].outputs[0].text

print(output)
```

```
<comparing two decimal numbers>
Since both numbers have the same integer part (9), we need to compare their fractional parts (0.11 and 0.9).
0.9 is greater than 0.11 because 9 is greater than 11.
</comparing two decimal numbers>
<answer>
9.9
</answer>
```

Llama 3 (trained with reasoning) produces correct output with reasoning steps