huggingface / open-r1 Public

Fully open reproduction of DeepSeek-R1

Apache-2.0 license

12.6k stars 947 forks Branches Tags Activity

Star Notifications

Code Issues 39 Pull requests 12 Discussions Actions Projects Security Insights

main 4 Branches 0 Tags Go to file Go to file Code

edbeeching Adds auto eval callbacks (#115) ✓ 972e47e · 39 minutes ago

.github	Bump actions/setup-python from 2 to 5 (#75)	3 days ago
assets	Add diagram (#16)	5 days ago
configs	Handle error in verifier + deepspeed command ...	5 days ago
scripts	Adds auto eval callbacks (#115)	39 minutes ago
slurm	Adds auto eval callbacks (#115)	39 minutes ago
src/open_r1	Adds auto eval callbacks (#115)	39 minutes ago
.gitignore	Adds auto eval callbacks (#115)	39 minutes ago
LICENSE	Initial commit	last week
Makefile	Implement make evaluate command (#41)	3 days ago
README.md	docs(README): note about CUDA 12.1 (#121)	1 hour ago
setup.cfg	Add configs and stuff (#2)	last week
setup.py	Improve repoduction of r1 reported score (#92)	yesterday

README Apache-2.0 license

Open R1

A fully open reproduction of DeepSeek-R1. This repo is a work in progress, let's build it together!

Overview

The goal of this repo is to build the missing pieces of the R1 pipeline such that everybody can reproduce and build on top of it. The project is simple by design and mostly consists of:

- `src/open_r1` : contains the scripts to train and evaluate models as well as generate synthetic data:
  - `grpo.py` : trains a model with GRPO on a given dataset.
  - `sft.py` : performs a simple SFT of a model on a dataset.
  - `evaluate.py` : evaluates a model on the R1 benchmarks.
  - `generate.py` : generates synthetic data from a model using Distilabel.
- `Makefile` : contains easy-to-run commands for each step in the R1 pipeline leveraging the scripts above.

Plan of attack

We will use the DeepSeek-R1 tech report as a guide, which can roughly be broken down into three main steps:

- Step 1: replicate the R1-Distill models by distilling a high-quality corpus from DeepSeek-R1.
- Step 2: replicate the pure RL pipeline that DeepSeek used to create R1-Zero. This will likely involve curating new, large-scale datasets for math, reasoning, and code.
- Step 3: show we can go from base model to RL-tuned via multi-stage training.

Installation

Note: Libraries rely on CUDA 12.1. Double check your system if you get segmentation faults.

To run the code in this project, first, create a Python virtual environment using e.g. `uv` . To install `uv` , follow the UV Installation Guide.

```
uv venv openr1 --python 3.11 && source openr1/bin/activate && uv pip install --upgrade pip
```

Next, install vLLM:

```
uv pip install vllm==0.6.6.post1

# For CUDA 12.1
pip install vllm==0.6.6.post1 --extra-index-url https://download.pytorch.org/whl/cu121
export LD_LIBRARY_PATH=$(python -c "import site; print(site.getsitepackages()[0] + '/nvidia/nvjitlink/lib')"):$LD_LIBRARY_PATH
```

This will also install PyTorch `v2.5.1` and it is **very important** to use this version since the vLLM binaries are compiled for it. You can then install the remaining dependencies for your specific use case via `pip install -e .[LIST OF MODES]` . For most contributors, we recommend:

```
pip install -e ".[dev]"
```

Next, log into your Hugging Face and Weights and Biases accounts as follows:

```
huggingface-cli login
wandb login
```

Finally, check whether your system has Git LFS installed so that you can load and push models/datasets to the Hugging Face Hub:

```
git-lfs --version
```

If it isn't installed, run:

```
sudo apt-get install git-lfs
```

Training models

We support training models with either DDP or DeepSpeed (ZeRO-2 and ZeRO-3). To switch between methods, simply change the path to the `accelerate` YAML config in `configs` .

> ⓘ Note
>
> The training commands below are configured for a node of 8 x H100s (80GB). For different hardware and topologies, you may need to tune the batch size and number of gradient accumulation steps.

SFT

To run SFT on a dataset distilled from DeepSeek-R1 with reasoning traces such as Bespoke-Stratos-17k, run:

```
accelerate launch --config_file=configs/zero3.yaml src/open_r1/sft.py \
    --model_name_or_path Qwen/Qwen2.5-Math-1.5B-Instruct \
    --dataset_name HuggingFaceH4/Bespoke-Stratos-17k \
    --learning_rate 2.0e-5 \
    --num_train_epochs 1 \
    --packing \
    --max_seq_length 4096 \
    --per_device_train_batch_size 4 \
    --per_device_eval_batch_size 4 \
    --gradient_accumulation_steps 4 \
    --gradient_checkpointing \
    --bf16 \
    --logging_steps 5 \
    --eval_strategy steps \
    --eval_steps 100 \
    --output_dir data/Qwen2.5-1.5B-Open-R1-Distill
```

To launch a Slurm job, run:

```
sbatch --output=/path/to/logs/%x-%j.out --err=/path/to/logs/%x-%j.err slurm/sft.slurm {model} {dataset} {accelerator}
```

Here `{model}` and `{dataset}` refer to the model and dataset IDs on the Hugging Face Hub, while `{accelerator}` refers to the choice of an 🤗 Accelerate config file in configs.

GRPO

```
accelerate launch --config_file configs/zero3.yaml src/open_r1/grpo.py \
    --output_dir DeepSeek-R1-Distill-Qwen-7B-GRPO \
    --model_name_or_path deepseek-ai/DeepSeek-R1-Distill-Qwen-7B \
    --dataset_name AI-MO/NuminaMath-TIR \
    --max_prompt_length 256 \
    --per_device_train_batch_size 1 \
    --gradient_accumulation_steps 16 \
    --logging_steps 10 \
    --bf16
```

Evaluating models

We use `lighteval` to evaluate models, with custom tasks defined in `src/open_r1/evaluate.py`. For models which fit on a single GPU, run:

```
MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
MODEL_ARGS="pretrained=$MODEL,dtype=float16,max_model_length=32768,gpu_memory_utilisation=0.8"
TASK=aime24
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "custom|$TASK|0|0" \
    --custom-tasks src/open_r1/evaluate.py \
    --use-chat-template \
    --system-prompt="Please reason step by step, and put your final answer within \boxed{}." \
    --output-dir $OUTPUT_DIR
```

To increase throughput across multiple GPUs, use *data parallel* as follows:

```
NUM_GPUS=8
MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
MODEL_ARGS="pretrained=$MODEL,dtype=float16,data_parallel_size=$NUM_GPUS,max_model_length=32768,gpu_memory_utilisation=0.8"
TASK=aime24
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "custom|$TASK|0|0" \
    --custom-tasks src/open_r1/evaluate.py \
    --use-chat-template \
    --system-prompt="Please reason step by step, and put your final answer within \boxed{}." \
    --output-dir $OUTPUT_DIR
```

For large models which require sharding across GPUs, use *tensor parallel* and run:

```
NUM_GPUS=8
MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-32B
MODEL_ARGS="pretrained=$MODEL,dtype=float16,tensor_parallel_size=$NUM_GPUS,max_model_length=32768,gpu_memory_utilisation=0.8"
TASK=aime24
OUTPUT_DIR=data/evals/$MODEL

export VLLM_WORKER_MULTIPROC_METHOD=spawn
lighteval vllm $MODEL_ARGS "custom|$TASK|0|0" \
    --custom-tasks src/open_r1/evaluate.py \
    --use-chat-template \
    --system-prompt="Please reason step by step, and put your final answer within \boxed{}." \
    --output-dir $OUTPUT_DIR
```

You can also launch an evaluation with `make evaluate`, specifying the model, task, and optionally the parallelism technique and number of GPUs.

To evaluate on a single GPU:

```
make evaluate MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-32B TASK=aime24
```

To use Data Parallelism:

```
make evaluate MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-32B TASK=aime24 PARALLEL=data NUM_GPUS=8
```

To use Tensor Parallelism:

```
make evaluate MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-32B TASK=aime24 PARALLEL=tensor NUM_GPUS=8
```

Reproducing Deepseek's evaluation results on MATH-500

We are able to reproduce Deepseek's reported results on the MATH-500 Benchmark:

Model	MATH-500 (HF lighteval)	MATH-500 (DeepSeek Reported)
DeepSeek-R1-Distill-Qwen-1.5B	81.6	83.9
DeepSeek-R1-Distill-Qwen-7B	91.8	92.8
DeepSeek-R1-Distill-Qwen-14B	94.2	93.9
DeepSeek-R1-Distill-Qwen-32B	95.0	94.3
DeepSeek-R1-Distill-Llama-8B	85.8	89.1
DeepSeek-R1-Distill-Llama-70B	93.4	94.5

To reproduce these results use the following command:

```
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B math_500
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Qwen-7B math_500
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Qwen-14B math_500
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Qwen-32B math_500 tp
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Llama-8B math_500
sbatch slurm/evaluate.slurm deepseek-ai/DeepSeek-R1-Distill-Llama-70B math_500 tp
```

Data generation

Generate data from a smol distilled R1 model

The following example can be run in 1xH100. First install the following dependencies:

```
uv pip install "distilabel[vllm]>=1.5.2"
```

Now save the following snippet into a file named `pipeline.py` and run it with `python pipeline.py`. It will generate 4 outputs for each of the 10 examples (change the username for the repository to your org/user name):

```
from datasets import load_dataset
from distilabel.models import vLLM
from distilabel.pipeline import Pipeline
from distilabel.steps.tasks import TextGeneration


prompt_template = """\
You will be given a problem. Please reason step by step, and put your final answer within \boxed{}:
{{ instruction }}"""

dataset = load_dataset("AI-MO/NuminaMath-TIR", split="train").select(range(10))

model_id = "deepseek-ai/DeepSeek-R1-Distill-Qwen-7B"  # Exchange with another smol distilled r1

with Pipeline(
    name="distill-qwen-7b-r1",
    description="A pipeline to generate data from a distilled r1 model",
) as pipeline:
    llm = vLLM(
        model=model_id,
        tokenizer=model_id,
        extra_kwargs={
            "tensor_parallel_size": 1,
            "max_model_len": 8192,
        },
        generation_kwargs={
            "temperature": 0.6,
            "max_new_tokens": 8192,
        },
    )
    prompt_column = "problem"
    text_generation = TextGeneration(
        llm=llm,
        template=prompt_template,
        num_generations=4,
        input_mappings={"instruction": prompt_column} if prompt_column is not None else {}
    )


if __name__ == "__main__":
    distiset = pipeline.run(dataset=dataset)
    distiset.push_to_hub(repo_id="username/numina-deepseek-r1-qwen-7b")
```

Take a look at the sample dataset at HuggingFaceH4/numina-deepseek-r1-qwen-7b.

Generate data from DeepSeek-R1

To run the bigger DeepSeek-R1, we used 2 nodes, each with 8×H100 GPUs using the slurm file present in this repo at `slurm/generate.slurm`. First, install the dependencies:

(for now we need to install the vllm dev wheel that fixes the R1 cuda graph capture)

```
pip install https://wheels.vllm.ai/221d388cc5a836fa189305785ed7e887cea8b510/vllm-1.0.0.dev-cp38-abi3-manylinux1_x86_64.whl --ex

uv pip install "distilabel[vllm,ray,openai]>=1.5.2"
```

And then run the following command:

```
sbatch slurm/generate.slurm \
    --hf-dataset AI-MO/NuminaMath-TIR \
    --temperature 0.6 \
    --prompt-column problem \
    --model deepseek-ai/DeepSeek-R1 \
    --hf-output-dataset username/r1-dataset
```

> ⓘ Note
>
> While the job is running, you can setup an SSH tunnel through the cluster login node to access the Ray dashboard from your computer running `ssh -L 8265:ray_ip_head_node:8265 <login_node>`, then browsing `http://localhost:8265`

Contributing

Contributions are welcome. Please refer to #23.

Releases

No releases published

Packages

No packages published

Contributors 22