Open R1

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

Table of Contents

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.

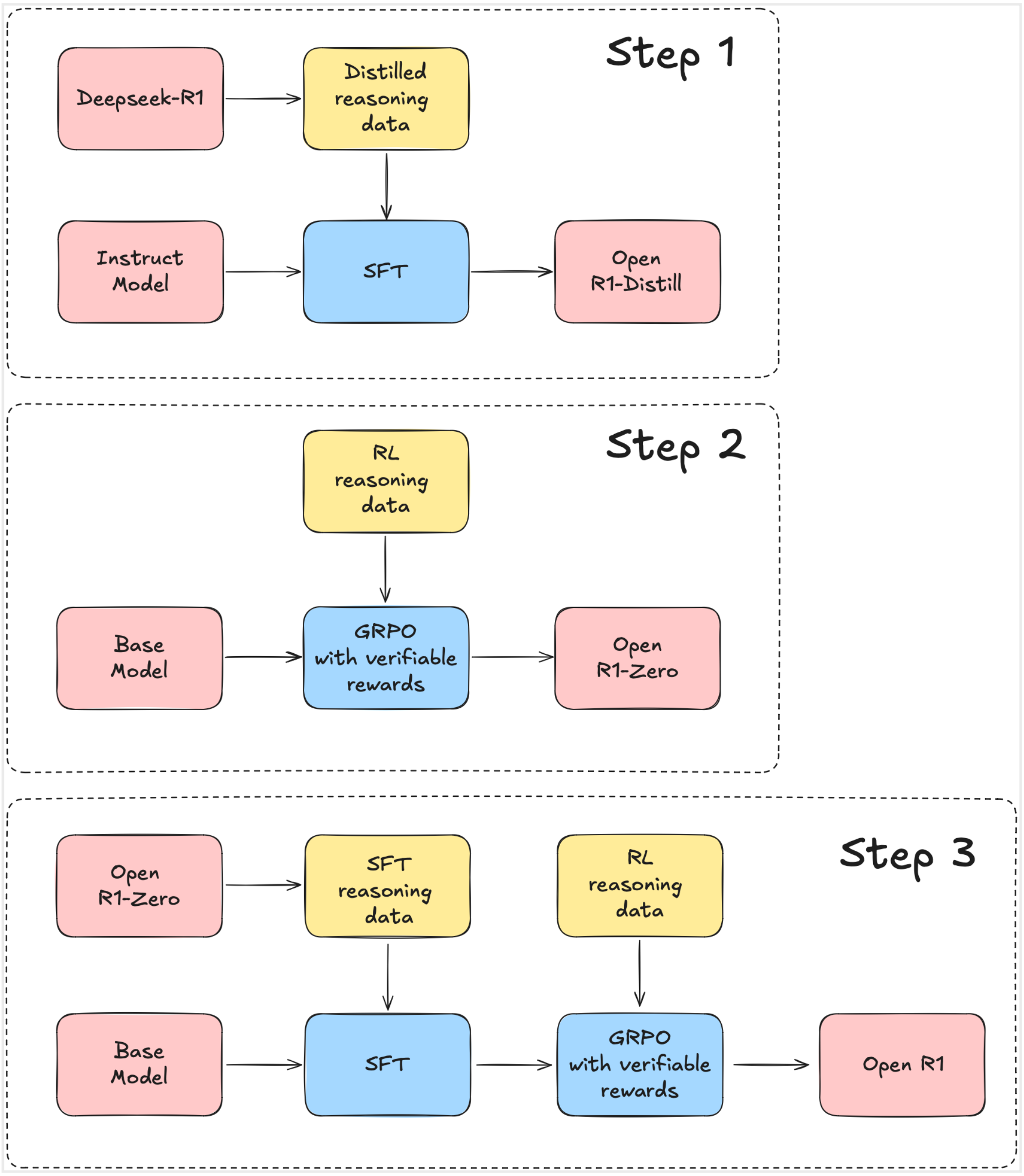
Plan of attack

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.



News 🗞️

- ⚡️ **[2025/03/11] (update #3):** We release the **CodeForces-CoTs** dataset of 10k competitive programming problems and 100k solutions distilled from R1. We also release IOI24: a new benchmark of *very* hard problems from international olympiads. A 7B Qwen model trained on CodeForces-CoTs can outperform Claude 3.7 Sonnet on IOI24, while a 32B model can outperform R1 itself.
- **∞ [2025/02/10] (update #2):** We release the **OpenR1-**

Math-220k dataset of 220k traces distilled from R1 on a new version of NuminaMath. Models trained on this dataset match the performance of DeepSeek's distilled ones.

- 🔥 **[2025/02/02] (update #1):** We implement the first parts of the training, inference, and evaluation pipelines. Let's go!

Installation

Caution

Libraries rely on CUDA 12.4. If you see errors related to segmentation faults, double check the version your system is running with nvcc --version.

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.

Note

As a shortcut, run make install to setup development libraries (spelled out below). Afterwards, if everything is setup correctly you can try out the Open-R1 models.

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

Tip

For Hugging Face cluster users, add export UV_LINK_MODE=copy to your .bashrc to suppress cache warnings from uv

Next, install vLLM and FlashAttention:

```
uv pip install vllm==0.8.3
uv pip install setuptools && uv pip install flash-attn --no-build-isolation
```

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:

```
GIT_LFS_SKIP_SMUDGE=1 uv 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

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.

We support training models with either DDP or DeepSpeed (ZeRO-2 and ZeRO-3). For example, to run SFT on a dataset distilled from DeepSeek-R1 with reasoning traces such as open-r1/OpenR1-Math-220k, run:

```
# Train via command line
accelerate launch --config_file=recipes/accelerate_configs/
zero3.yaml src/open_r1/sft.py \
    --model_name_or_path Qwen/Qwen2.5-1.5B-Instruct \
    --dataset_name open-r1/OpenR1-Math-220k \
    --learning_rate 5.0e-5 \
    --num_train_epochs 1 \
    --max_seq_length 16384 \
    --per_device_train_batch_size 16 \
    --gradient_checkpointing \
    --bf16 \
    --use_liger_kernel \
    --output_dir data/Qwen2.5-1.5B-Open-R1-Distill

# Train via YAML config
accelerate launch --config_file recipes/accelerate_configs/zero3.yaml
src/open_r1/sft.py \
    --config recipes/Qwen2.5-1.5B-Instruct/sft/config_demo.yaml
```

Currently, the following tasks are supported:

- Supervised Fine-Tuning sft
- Group Relative Policy Optimization grpo

Tip

If you scale up/down the number of GPUs, we recommend also scaling up the per-device batch size or number of gradient accumulation steps to keep the global batch size constant.

By default, these scripts will push each model to your Hugging Face Hub username, i.e. {username}/{model_name}-{task}. You can override the parameters in each YAML config by appending them to the command as follows:

```
# Change batch size, number of epochs etc
accelerate launch --config_file recipes/accelerate_configs/zero3.yaml
src/open_r1/sft.py \
    --config recipes/Qwen2.5-1.5B-Instruct/sft/config_demo.yaml
    --per_device_train_batch_size=1 --num_train_epochs=5
```

If you also wish to override the Weights and Biases default settings, you can do so as follows:

```
accelerate launch --config_file recipes/accelerate_configs/zero3.yaml
src/open_r1/sft.py \
    --config recipes/Qwen2.5-1.5B-Instruct/sft/config_demo.yaml
    --wandb_entity huggingface --wandb_project open-r1 --run_name
Qwen2.5-1.5B-GRPO
```

🚨 WARNING 🚨

Most base models like meta-llama/Llama-3.2-1B do not have a chat template, so we set ChatML as the default during training. However, for Qwen base models

like Qwen/Qwen2.5-1.5B, a chat template is pre-defined in the tokenizer, so the EOS token must be set accordingly, e.g.

```
# Align EOS token with chat template for Qwen base models
accelerate launch --config_file=recipes/accelerate_configs/
zero3.yaml src/open_r1/sft.py \
   --model_name_or_path Qwen/Qwen2.5-1.5B \
+  --eos_token '<|im_end|>'
   --dataset_name open-r1/OpenR1-Math-220k \
   --learning_rate 5.0e-5 \
   --num_train_epochs 1 \
   --max_seq_length 16384 \
   --per_device_train_batch_size 16 \
   --gradient_checkpointing \
   --bf16 \
   --use_liger_kernel \
   --output_dir data/Qwen2.5-1.5B-Open-R1-Distill
```

If you wish to use a custom chat template (e.g. Llama or Gemma), then the chat template and associated EOS token must be provided:

```
# Align EOS token with custom chat template
accelerate launch --config_file=recipes/accelerate_configs/
zero3.yaml src/open_r1/sft.py \
   --model_name_or_path meta-llama/Llama-3.2-1B \
+  --chat_template "$(cat llama_chat_template.jinja)" \
+  --eos_token '<|eot_id|>' \
   --dataset_name open-r1/OpenR1-Math-220k \
   --learning_rate 5.0e-5 \
   --num_train_epochs 1 \
   --max_seq_length 16384 \
   --per_device_train_batch_size 16 \
   --gradient_checkpointing \
   --bf16 \
   --use_liger_kernel \
   --output_dir data/Llama-3.2-1B-Open-R1-Distill
```

SFT

To run SFT on a dataset distilled from DeepSeek-R1 with reasoning traces such as open-r1/OpenR1-Math-220k,

run:

```
ACCELERATE_LOG_LEVEL=info accelerate launch --config_file
recipes/accelerate_configs/zero3.yaml \
    src/open_r1/sft.py \
    --config recipes/Qwen2.5-1.5B-Instruct/sft/config_demo.yaml
```

GRPO

We use TRL's vLLM backend to scale training to large models across multiple nodes. For single-node training of smol models across 8 GPUs, first spin up the vLLM server to run on e.g. 1 GPU as follows:

```
CUDA_VISIBLE_DEVICES=0 trl vllm-serve --model deepseek-ai/
DeepSeek-R1-Distill-Qwen-1.5B
```

Once the server is up, run training on the remaining GPUs as follows:

```
CUDA_VISIBLE_DEVICES=1,2,3,4,5,6,7
ACCELERATE_LOG_LEVEL=info \
    accelerate launch --config_file recipes/accelerate_configs/
zero2.yaml --num_processes 7 \
    src/open_r1/grpo.py --config recipes/DeepSeek-R1-Distill-
Qwen-1.5B/grpo/config_demo.yaml
```

Warning

The chat template used in the distilled DeepSeek models omits the contents of the reasoning block within the <think> and </think> tags. It also prefills the assistant response with <think> which interferes with the format reward function. To handle that, it is important to override the chat template as done in e.g. recipes/DeepSeek-R1-Distill-Qwen-1.5B/grpo/config_demo.yaml.

For multi-node training, we provide an example Slurm script:

```
sbatch --nodes=2 slurm/train.slurm Qwen2.5-Math-7B grpo
config_simple_rl zero3
```

You will need to adapt the slurm/train.slurm script to

match your cluster.

🧑‍💻 Training with a code interpreter

We provide a code reward function for executing code generated by the policy during training. Currently, this reward function targets code contests like Codeforces, where solutions are executed against a set of test cases and the overall success rate is returned as the final reward. To ensure safe execution, we use E2B sandboxes, which are fast and cheap to run. To use this reward function, first install the necessary dependencies:

```
uv pip install -e '.[code]'
```

Then create a .env file and place an API token from E2B within it:

```
E2B_API_KEY="e2b_xxx"
```

Then make sure your dataset contains a verification_info column with the following schema (adopted from PrimeIntellect's excellent datasets of verifiable problems):

```
{
    "language": "python",
    "test_cases": [
        {
            "input":
"4\n4\n0001\n1000\n0011\n0111\n3\n010\n101\n0\n2\n00000\n00001
\n4\n01\n001\n0001\n00001\n",
            "output": "1\n3 \n-1\n0\n\n2\n1 2 \n",
            "type": "stdin_stdout",
        }
    ],
}
```

For example, to train a smol model on Python problems, start the vLLM server:

```
CUDA_VISIBLE_DEVICES=0 trl vllm-serve --model Qwen/
Qwen2.5-1.5B-Instruct
```

Then run training with:

```
CUDA_VISIBLE_DEVICES=1,2,3,4,5,6,7
ACCELERATE_LOG_LEVEL=info \
    accelerate launch --config_file recipes/accelerate_configs/
zero2.yaml --num_processes=7 \
    src/open_r1/grpo.py --config recipes/Qwen2.5-1.5B-Instruct/grpo/
config_demo_code.yaml
```

It is possible to be rate limited when too many scripts are executed on E2B Sandboxes, so we provide an E2B router script that can be launched on a CPU node on your cluster:

For GRPO training: First start the router and get its IP

```
sbatch slurm/e2b_router.slurm
```

Then add this line in your training YAML config: (for example)

```
e2b_router_url: 1.2.3.4:8000
```

The port here should match the one used when launching the router. All training jobs can share the same router IP which will ensure there are at most 20 parallel executions.

IOI problems

We provide a ioi_code_reward reward function for executing problems from IOI using piston.

To get piston workers running, see slurm/piston/README.md. Set your environment variable PISTON_ENDPOINTS to slurm or to a list of piston worker endpoints.

See the example recipe for how to use the reward function:

```
ACCELERATE_LOG_LEVEL=info accelerate launch --config_file
recipes/accelerate_configs/zero2.yaml \
    --num_processes=7 src/open_r1/grpo.py \
    --config recipes/Qwen2.5-1.5B-Instruct/grpo/
config_demo_code_ioi.yaml
```

Data decontamination

Following s1: Simple test-time scaling the data can be decontaminated using the script at: scripts/decontaminate.py, which decontaminates a dataset using 8-grams and deduplicate the data. Sample run:

```
python scripts/decontaminate.py \
    --dataset "open-r1/verifiable-coding-problems-python" \
    --problem_column problem \
    --cleanup
```

It will decontaminate against the benchmark datasets, and remove the contaminated samples afterwards. If no argument --new_dataset_name is provided, the same dataset will be reused, adding a _decontaminated. It runs against the prompt, which for this dataset is the column problem, but a different one can be provided.

Arguments for the script:

```
usage: decontaminate.py [-h] --dataset DATASET [--split SPLIT] [--ngram_size NGRAM_SIZE] [--problem_column PROBLEM_COLUMN] [--cleanup] [--new_dataset_name NEW_DATASET_NAME]

options:
  -h, --help            show this help message and exit
  --dataset DATASET     Name of the dataset to check for contamination.
  --split SPLIT         Split to check for contamination, defaults to `train`.
  --ngram_size NGRAM_SIZE
                        Size of n-grams to build, defaults to 8.
  --problem_column PROBLEM_COLUMN
                        Name of the column containing the problem (prompt).
  --cleanup             Whether to remove the contaminated rows before pushing the dataset.
  --new_dataset_name NEW_DATASET_NAME
                        New name for the dataset. If not provided, will reuse the name and add a `_decontaminated` to the name.
```

Launching jobs on a Slurm cluster

If you have access to a Slurm cluster, we provide a slurm/train.slurm script that will automatically queue training jobs for you. Here's how you can use it:

```
sbatch --job-name=open_r1 --nodes=1 slurm/train.slurm
{model_name} {task} {config_suffix} {accelerator}
```

Here {model_name} and {task} are defined as above, while {config_suffix} refers to the specific config and {accelerator} refers to the choice of 🤗 Accelerate config in recipes/accelerate_configs. If you wish to override the default config parameters, you can provide them by appending a space-separated string like '--arg1=value1 --arg2=value2'. Here's a concrete example to run SFT on 1 node of 8 GPUs:

```
# Launch on Slurm and override default hyperparameters
sbatch --job-name=open_r1 --nodes=1 slurm/train.slurm
Qwen2.5-1.5B-Instruct sft demo zero3 '--
per_device_train_batch_size=1 --num_train_epochs=5'
```

You can scale the number of nodes by increasing the --nodes flag.

Note
The configuration in slurm/train.slurm is optimised for the Hugging Face Compute Cluster and may require tweaking to be adapted to your own compute nodes.

Evaluating models

We use lighteval to evaluate models. For models which fit on a single GPU, run:

```
MODEL=deepseek-ai/DeepSeek-R1-Distill-Qwen-1.5B
MODEL_ARGS="pretrained=$MODEL,dtype=bfloat16,max_model_len
gth=32768,max_num_batched_tokens=32768,gpu_memory_utilizati
on=0.8,generation_parameters={max_new_tokens:32768,temperatur
e:0.6,top_p:0.95}"
OUTPUT_DIR=data/evals/$MODEL

# AIME 2024
```

```
TASK=aime24
lighteval vllm $MODEL_ARGS "lighteval|$TASK|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR

# MATH-500
TASK=math_500
lighteval vllm $MODEL_ARGS "lighteval|$TASK|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR

# GPQA Diamond
TASK=gpqa:diamond
lighteval vllm $MODEL_ARGS "lighteval|$TASK|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR

# LiveCodeBench
lighteval vllm $MODEL_ARGS "extended|lcb:codegeneration|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR
```

Important
You must
set max_model_length=32768 and max_num_batched_tokens=32768 in the vllm command to align with the max_new_tokens we define per eval. Without this, lighteval will throw an error.
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=$MODEL,dtype=bfloat16,data_parallel_size=$NUM_GPUS,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
TASK=aime24
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "lighteval|$TASK|0|0" \
    --use-chat-template \
    --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=$MODEL,dtype=bfloat16,tensor_parallel_size=$NUM_GPUS,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
TASK=aime24
OUTPUT_DIR=data/evals/$MODEL

export VLLM_WORKER_MULTIPROC_METHOD=spawn
lighteval vllm $MODEL_ARGS "lighteval|$TASK|0|0" \
    --use-chat-template \
    --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

Note

The DeepSeek-R1 paper uses sampling with 4-64 responses per query to estimate pass@1 accuracy, but

does not specify the specific number of responses per benchmark. For AIME 2024, we report the results from sampling 32 response per query, while for all others we report the accuracy from sampling 1 response. These choices likely explains the small 1-3σ discrepancies between our results and DeepSeek's.

AIME 2024

We are able to reproduce Deepseek's reported results on the AIME 2024 benchmark within ~1-3 standard deviations:

Model	AIME 2024 (🤗 LightEval)	AIME 2024 (DeepSeek Reported)
DeepSeek-R1-Distill-Qwen-1.5B	31.8	28.9
DeepSeek-R1-Distill-Qwen-7B	52.2	55.5
DeepSeek-R1-Distill-Qwen-14B	66.5	69.7
DeepSeek-R1-Distill-Qwen-32B	68.0	72.6
DeepSeek-R1-Distill-Llama-8B	43.9	41.7
DeepSeek-R1-Distill-Llama-70B	65.3	70.0

To reproduce these results use the following command:

```
NUM_GPUS=1 # Set to 8 for 32B and 70B models
MODEL=deepseek-ai/{model_name}
MODEL_ARGS="pretrained=$MODEL,dtype=bfloat16,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,data_parallel_size=$NUM_GPUS,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "lighteval|aime24|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR
```

Alternatively, you can launch Slurm jobs as follows:

```
python scripts/run_benchmarks.py --model-id {model_id}  --benchmarks aime24
```

MATH-500

We are able to reproduce Deepseek's reported results on the MATH-500 benchmark within ~1-3 standard deviations:

To reproduce these results use the following command:

```
NUM_GPUS=1 # Set to 8 for 32B and 70B models
MODEL=deepseek-ai/{model_name}
MODEL_ARGS="pretrained=$MODEL,dtype=bfloat16,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,data_parallel_size=$NUM_GPUS,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "lighteval|math_500|0|0" \
   --use-chat-template \
   --output-dir $OUTPUT_DIR
```

Alternatively, you can launch Slurm jobs as follows:

```
python scripts/run_benchmarks.py --model-id {model_id}  --benchmarks math_500
```

GPQA Diamond

We are able to reproduce Deepseek's reported results on the GPQA Diamond benchmark within ~1-3 standard deviations:

To reproduce these results use the following command:

```
NUM_GPUS=1 # Set to 8 for 32B and 70B models
MODEL=deepseek-ai/{model_name}
MODEL_ARGS="pretrained=$MODEL,dtype=bfloat16,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,data_parallel_size=$NUM_GPUS,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "lighteval|gpqa:diamond|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR

python scripts/run_benchmarks.py --model-id {model_id}  --benchmarks gpqa
```

LiveCodeBench

We are able to reproduce Deepseek's reported results on the LiveCodeBench code generation benchmark within ~1-3 standard deviations:

To reproduce these results use the following command:

```
NUM_GPUS=1 # Set to 8 for 32B and 70B models, or data_parallel_size=8 with the smaller models for speed
MODEL=deepseek-ai/{model_name}
MODEL_ARGS="pretrained=$MODEL,dtype=bfloat16,max_model_length=32768,max_num_batched_tokens=32768,gpu_memory_utilization=0.8,data_parallel_size=$NUM_GPUS,generation_parameters={max_new_tokens:32768,temperature:0.6,top_p:0.95}"
```

```
OUTPUT_DIR=data/evals/$MODEL

lighteval vllm $MODEL_ARGS "extended|lcb:codegeneration|0|0" \
    --use-chat-template \
    --output-dir $OUTPUT_DIR

python scripts/run_benchmarks.py --model-id {model_id}  --
benchmarks lcb
```

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 --extra-index-url https://download.pytorch.org/whl/cu121

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.

Acknowledgements

This project is built with the collective efforts of many groups and individuals in the open AI community. We are especially grateful to the vLLM and SGLang teams for creating high-performance tooling to scale the rollouts of GRPO. We also thank the teams at OpenThoughts, Prime Intellect, and General Reasoning for creating and sharing high-quality datasets for reasoning.

Citation

If you find this project is useful in your own work, please consider citing as follows:

```
@misc{openr1,
    title = {Open R1: A fully open reproduction of DeepSeek-R1},
    url = {https://github.com/huggingface/open-r1},
    author = {Hugging Face},
    month = {January},
    year = {2025}
}
```