Guest Lecture: Self-Distillation



Idan Shenfeld

Idan Shenfeld is a Member of Technical Staff at Thinking Machines Lab. He did his PhD at MIT with Pulkit Agrawal in the Improbable AI Lab, where he worked on reinforcement learning for language models, and in particular on how a model can keep learning new skills without forgetting what it already knows. His paper RL's Razor (ICLR 2026) showed that on-policy RL forgets less than supervised fine-tuning because it stays closer to the original model. The follow-up, Self-Distillation Enables Continual Learning (ICML 2026; Best Paper, ICLR Lifelong Agents Workshop), turned this into a method that needs no reward: given a demonstration in its prompt, the model becomes its own teacher. Cursor has since used self-distillation to train Composer 2.5. During his PhD, he also worked on LLM post-training at Google DeepMind.

Web: https://www.idanshenfeld.com/ https://www.linkedin.com/in/idan-shenfeld
Blog: https://cursor.com/blog/composer-2-5

multisensory intelligence

How to AI an AI for Almost Anything

Lecture 6 - RL & Self-Distillation in Practice

Ao Qu

mit media lab

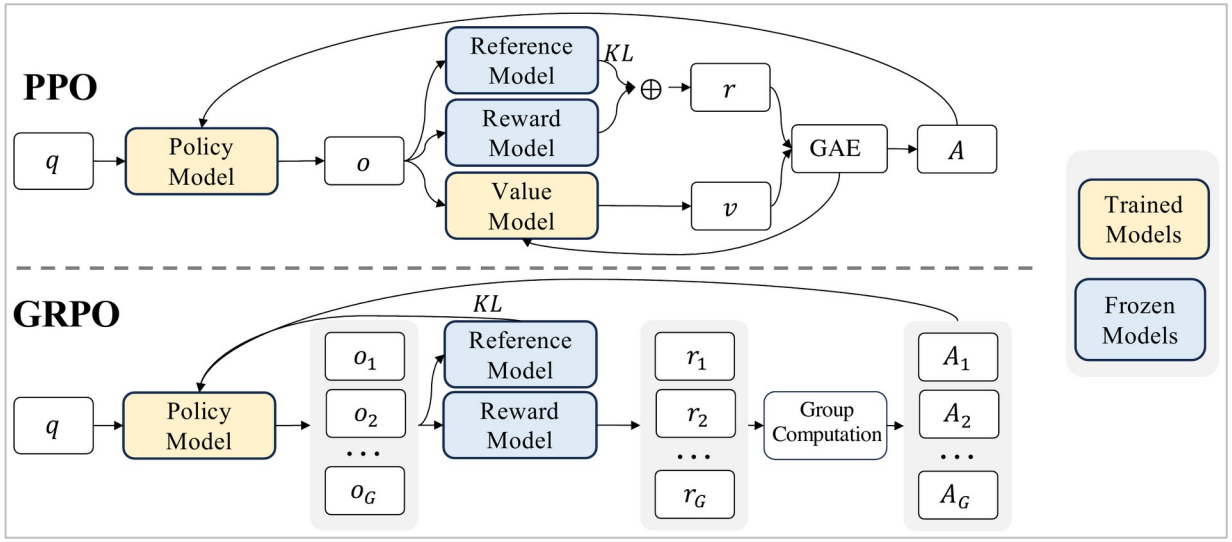
https://quao627.github.io/
https://x.com/ao_qu18465

MIT EECS

Today's Tutorial

- Recap: PPO & GRPO, OPD & self-distillation
- What the RL loop looks like
- RL vs OPD (on-policy distillation)
- What to watch during training
- verl quickstart
- From verl to reef: infra for self-evolving agents

Recap: RL for LLMs: PPO and GRPO



An LLM as a policy: each token is an action

prompt **x** = "What is 5 + (2 × 3)?"

action $y_t \sim \pi_\theta(\cdot|s_t)$

x	5	+	2	is	7	,	…	21	.	r = 0

state $s_t = (x, y_{<t})$

reward: once, at the end

a **wrong** answer (21 ≠ 11) → r = 0

policy-gradient loss: $L(\theta) = -\Sigma_t A_t \log \pi_\theta(y_t \mid s_t)$

PPO / GRPO in practice: $L(\theta) = -\Sigma_t \min(\rho_t A_t,\ \text{clip}(\rho_t, 1-\varepsilon, 1+\varepsilon) A_t),\quad \rho_t = \pi_\theta(y_t|s_t) / \pi_{old}(y_t|s_t)$

PPO: $A_t = Q(s_t, y_t) - V(s_t)$

V = a learned value model (GAE)

GRPO: $\hat{A}_i = (r_i - \text{mean}(r)) / \text{std}(r)$

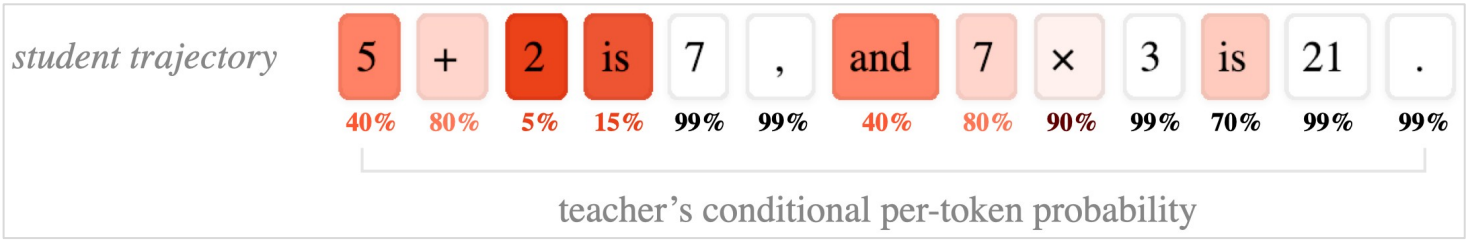
over G answers to the same prompt; no V

[Shao et al., DeepSeekMath 2024] (figure) [Schulman et al., PPO 2017] [Schulman et al., GAE. ICLR 2016]
Slides: https://mit-mi.github.io/how2ai-course/fall2026/schedule/lec6%20-%20training.pdf

multisensory intelligence

Recap: On-Policy Distillation and Self-Distillation

prompt: “What is 5 + (2 × 3)?”



① student writes $y \sim \pi_\theta$

② teacher scores **every token**

③ student moves toward it

reverse KL per token: $KL(\pi_\theta \| \pi_T) = E_{y_t \sim \pi_\theta}[\log \pi_\theta(y_t|s_t) - \log \pi_T(y_t|s_t)]$

gradient: $\nabla KL = -E_{y_t \sim \pi_\theta}[A_t \nabla \log \pi_\theta(y_t|s_t)]$, $\mathbf{A_t = \log \pi_T(y_t|s_t) - \log \pi_\theta(y_t|s_t)}$ (as $E[\nabla \log \pi_\theta] = 0$)

→ a policy-gradient loss: $L(\theta) = -\Sigma_t A_t \log \pi_\theta(y_t|s_t)$

OPD: the teacher is **another, bigger model** π_T

teacher reads x y<t → $\pi_T(y_t \mid x, y_{<t})$

$A_t = \log \pi_T(y_t|s_t) - \log \pi_\theta(y_t|s_t)$

e.g. Qwen3-32B teaching Qwen3-8B

self-distillation: the teacher is **the model itself + c**

teacher reads c x y<t → $\text{sg}[\pi_\theta(y_t \mid \mathbf{c}, x, y_{<t})]$

$A_t = \log \pi_\theta(y_t|\mathbf{c}, s_t) - \log \pi_\theta(y_t|s_t)$

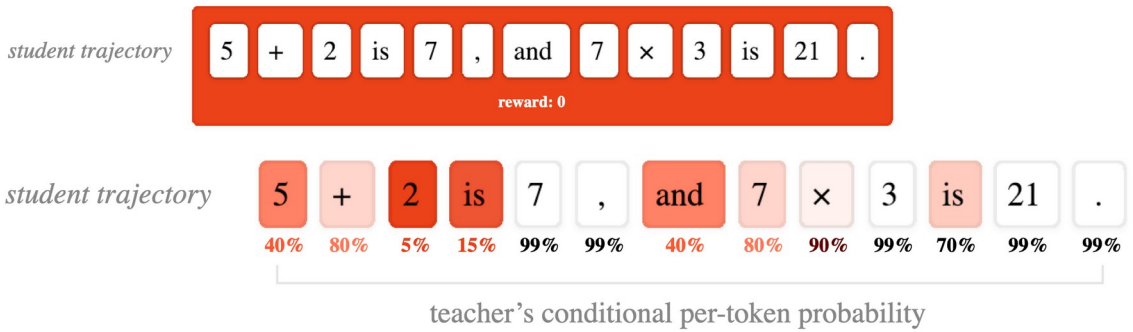
c = a demo (SDFT), the solution (OPSD), feedback (SDPO)

[Agarwal et al., GKD. ICLR 2024] [Shenfeld et al., SDFT 2026] [Zhao et al., OPSD 2026] [Hübotter et al., SDPO 2026]
Blog: https://thinkingmachines.ai/blog/on-policy-distillation/ (figure)

multisensory intelligence

RL vs OPD

same loss: $L(\theta) = -\Sigma_t A_t \log \pi_\theta(y_t \mid s_t)$



RL $A_t = r - \text{baseline}$
→ **one number** for all tokens

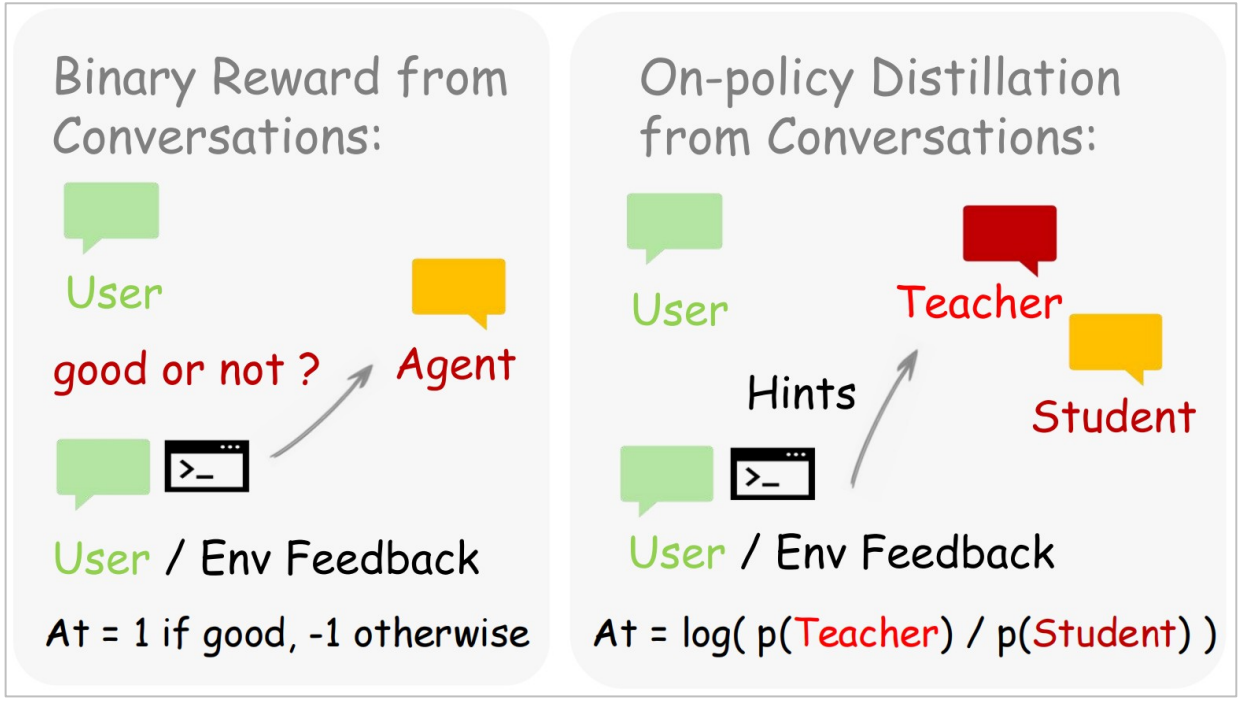
OPD $A_t = \log \pi_T(y_t|s_t) - \log \pi_\theta(y_t|s_t)$
→ **one number per token**

	RL	OPD
information	O(1) bits per episode	O(N) bits, N = tokens
learns toward	high reward (can be hacked)	the teacher (its ceiling)

[Lu et al., On-Policy Distillation. Thinking Machines Lab 2025] Blog: https://thinkingmachines.ai/blog/on-policy-distillation/ (figures)
[Shao et al., DeepSeekMath 2024] [Agarwal et al., GKD. ICLR 2024]

multisensory intelligence

OpenClaw-RL: Training on Human Feedback



example: a TA uses the agent to grade homework

agent: Correct answer: 220. Your solution is correct.

user: Too brief. Tell the student why each step is right.

① **RL:** a judge reads the reply: good or not?
→ r = −1: the **same** A for every token of the answer

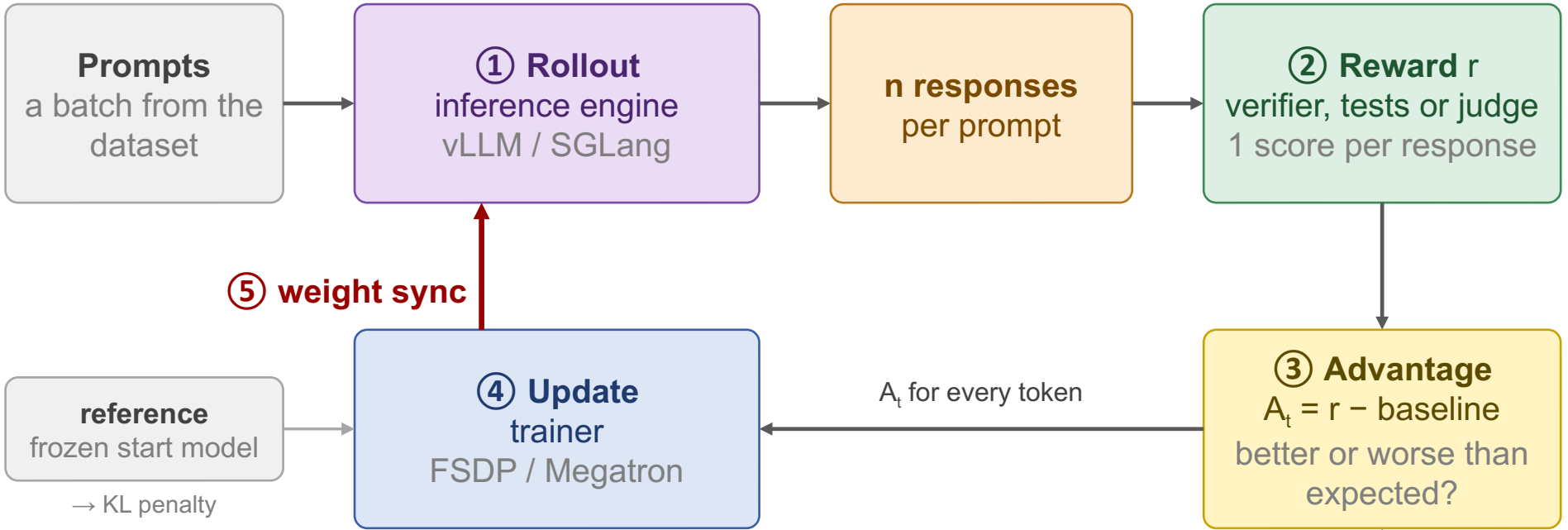
② **Self-distillation:** the judge turns the reply into a hint h ("explain each step"); the same model with h in its prompt is the teacher
→ $A_t = \log \pi_\theta(y_t \mid h) - \log \pi_\theta(y_t)$, **per token**

loss = ① + ②, on the same responses, in one update
① a score on every turn, but one number; ② which tokens to change, but only when the reply has a correction

[Wang et al., OpenClaw-RL: Train Any Agent Simply by Talking. arXiv 2026]
Code: https://github.com/Gen-Verse/OpenClaw-RL Blog: https://yinjjiew.github.io/projects/openclawrl1 (figure)

multisensory intelligence

One Training Step



Q: Why does the model live in **two places**?

A: generation needs a fast KV-cache engine, training needs gradients and optimizer state → copy the weights after every update

[Sheng et al., HybridFlow: A Flexible and Efficient RLHF Framework. EuroSys 2025]
Docs: https://verl.readthedocs.io/

multisensory intelligence

Rollout vs Trainer

① **Rollout**: inference engine
vLLM / SGLang

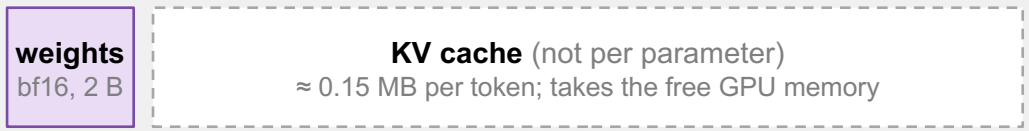
job: predict the next token, again and again, until the response is done

forward pass only: **no gradients**

keeps a **KV cache**, so past tokens are not recomputed

vLLM: KV cache in pages (PagedAttention) + continuous batching → many responses at once

memory per parameter (Qwen3-8B, 8.2B params)



2 bytes × 8.2B ≈ **16 GB** + KV cache

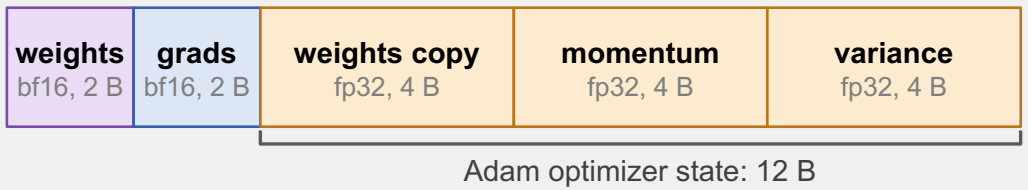
④ **Update**: trainer
FSDP / Megatron

job: gradients of the loss over whole responses

forward + **backward**: keeps activations, gradients and the optimizer state

FSDP: splits weights, gradients and optimizer state across GPUs; gathers a layer's full weights just before using it

memory per parameter (same scale)



16 bytes × 8.2B ≈ **130 GB** + activations; > one 80 GB GPU

[Kwon et al., PagedAttention (vLLM). SOSP 2023] [Zhao et al., PyTorch FSDP. VLDB 2023] [Rajbhandari et al., ZeRO. SC 2020]

multisensory intelligence

Vocabulary

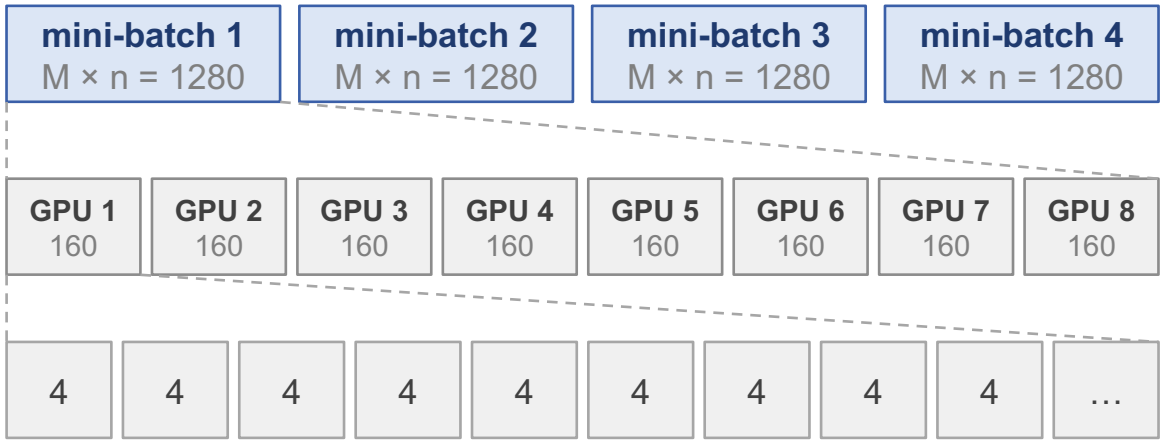
term	symbol	what it is	in verl
policy / actor	π_θ	the model being trained	`actor`
rollout		sampling responses with the current model	`rollout`
state	s_t	the prompt + the tokens so far	
reward	r	score of one response	`token_level_scores`
advantage	A_t	better or worse than expected, per token	`advantages`
old policy	π_{old}	the weights that sampled this batch	`old_log_probs`
reference policy	π_{ref}	the frozen start model	`ref_log_prob`
critic / value model	$V(s_t)$	predicts the reward from s_t (PPO only)	`critic`, `values`
step		one rollout + update	`global_steps`
epoch		one pass over the prompt dataset	`total_epochs`

Docs: https://verl.readthedocs.io/ Code: https://github.com/verl-project/verl/blob/ddf7007/verl/trainer/ppo/v1/trainer_base.py

multisensory intelligence

Batch → Mini-batch → Micro-batch

training batch = **B** × **n** = 5120 responses



GPU 1's 160 responses, **m** = 4 at a time

B = `train_batch_size` = 1024 prompts
n = `rollout.n` = 5 responses per prompt
→ B × n = **5120** responses = one rollout

M = `ppo_mini_batch_size` = 256 prompts
→ B ÷ M = **4** optimizer steps
mini-batch: the data for one weight update

G = 8 GPUs (data parallel)
→ M × n ÷ G = **160** responses per GPU

m = `ppo_micro_batch_size_per_gpu` = 4
→ 160 ÷ m = **40** micro-batches = **grad accumulation**
micro-batch: split only so it fits in GPU memory

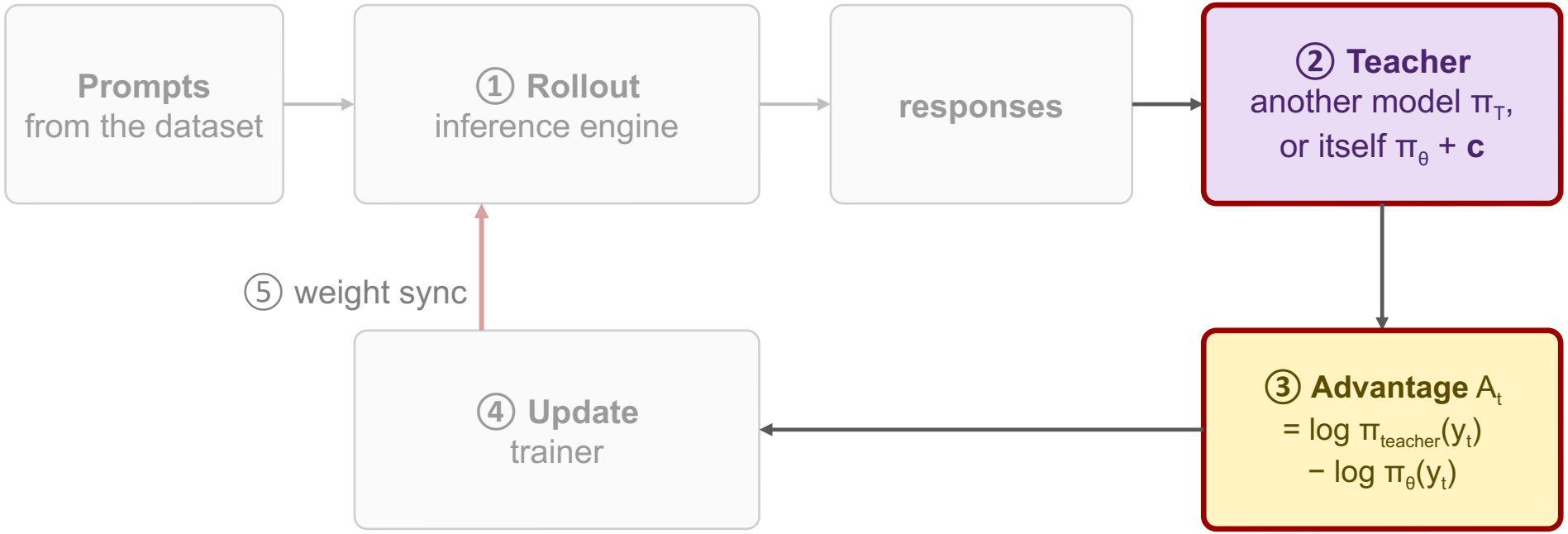
Q: On-policy = trained on samples from the current weights. Mini-batch 1 or 2: which is on-policy?

A: 1 only. With more than one mini-batch, the model is **updated** after each one, so from **2** on it is no longer the policy that sampled the rollout: $\pi_\theta \neq \pi_{old}$.

Docs: https://verl.readthedocs.io/en/latest/algo/grpo.html#configuration (figure of the batch sizes)
Code: https://github.com/verl-project/verl/blob/ddf7007/examples/grpo_trainer/run_qwen3_8b_fsdp.sh (the numbers above)

multisensory intelligence

On-Policy Distillation and Self-Distillation



	on-policy distillation (OPD)	self-distillation
teacher	**another, bigger model**	the model **itself**, with **c** in its prompt
knows more from	its size and training	**c** only: without it, teacher = student → no signal

[Agarwal et al., GKD 2024] [Shenfeld et al., SDFT 2026] [Zhao et al., OPSD 2026] [Hübotter et al., SDPO 2026] [Kaur et al., 2026]
Blog: https://thinkingmachines.ai/blog/on-policy-distillation/

multisensory intelligence

RL vs Distillation

	method	who scores a response	advantage A_t of token y_t
RL	**PPO**	reward r(x, y)	$Q(s_t, y_t) - V(s_t)$ V = a learned critic
	GRPO	reward r(x, y)	(r − mean) / std over the n responses to a prompt: the mean replaces V
distillation	**on-policy (OPD)**	**another**, bigger model π_T	$\log \pi_T(y_t) - \log \pi_\theta(y_t)$ a different value on every token
	self-distillation	the **same model** π_θ + **c**	$\log \pi_\theta(y_t \mid \mathbf{c}) - \log \pi_\theta(y_t)$ only difference: **c** in the teacher's prompt

$V(s_t)$: reward expected from here, where s_t = the prompt + the tokens so far

$Q(s_t, y_t)$: reward expected after picking token y_t

[Schulman et al., PPO 2017] [Schulman et al., GAE. ICLR 2016] [Shao et al., DeepSeekMath 2024] [Agarwal et al., GKD 2024]
Blog: https://thinkingmachines.ai/blog/on-policy-distillation/

multisensory intelligence

Entropy and KL in the Loss

$$-\min(\ \rho_t A_t\ ,\ \ \mathrm{clip}(\rho_t,\ 1-\varepsilon,\ 1+\varepsilon)\ A_t\ )$$

policy gradient

$\rho_t = \pi_\theta / \pi_{old}$: how much the token's probability changed

$A_t > 0$ pushes the token up, < 0 down

the clip keeps each step small

$$+\ \beta\ \mathrm{KL}(\pi_\theta \,\|\, \pi_{ref})$$

KL to the frozen model

π_{ref} = the **frozen start model**

keeps the model close to it

reasoning RL: usually **off**

RLHF: **on**

$$-\ \alpha\ H(\pi_\theta)$$

entropy

how random the next token is

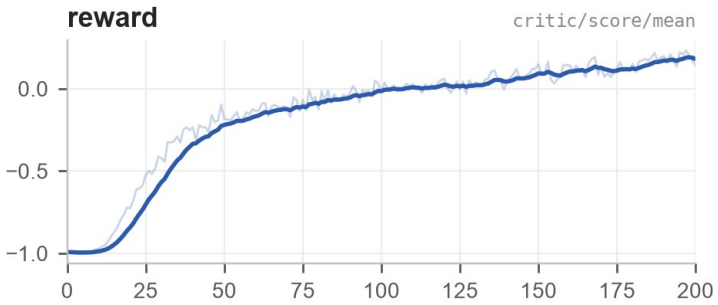
near 0 = stopped exploring

the bonus is usually off, but always watch the curve

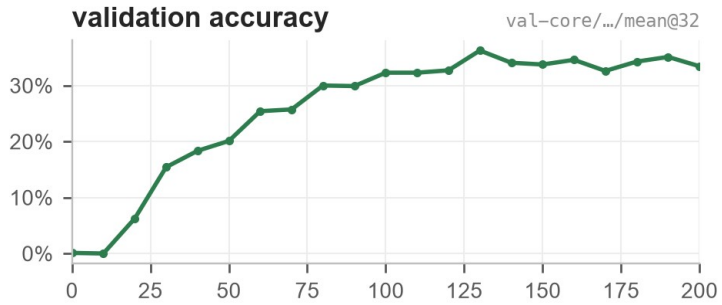
[Schulman et al., Proximal Policy Optimization Algorithms 2017] [Shao et al., DeepSeekMath 2024] [Yu et al., DAPO 2025]
Code: https://github.com/verl-project/verl/blob/ddf7007/verl/trainer/ppo/core_algos.py

multisensory intelligence

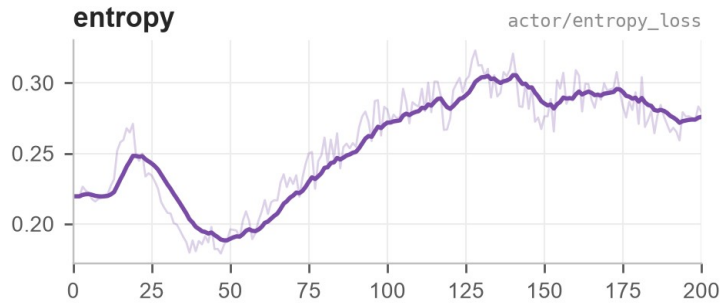
An Example Run: Reasoning RL on Math



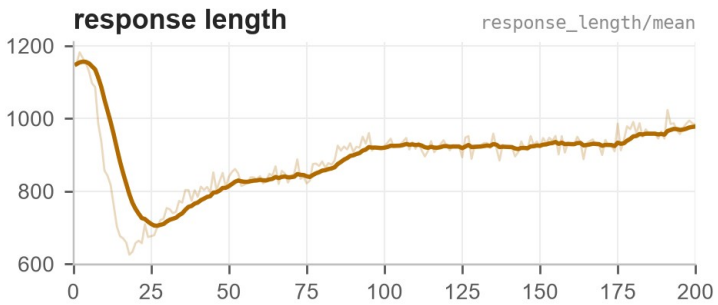
normal: slow start, then a steady climb
red flag: flat from the start: no signal



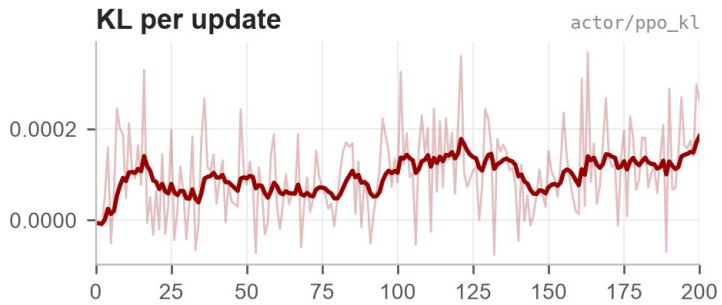
normal: rises along with the reward
red flag: reward up, validation down



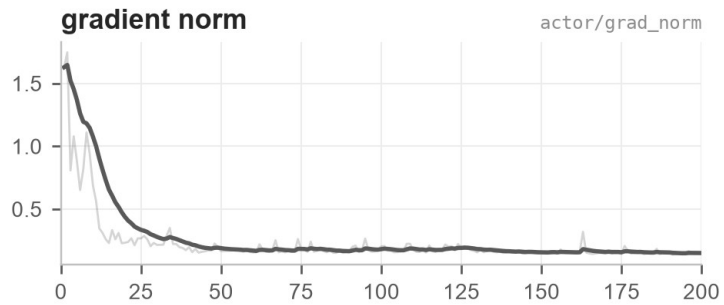
normal: stays well above 0
red flag: falls toward 0: collapse



normal: changes as the reasoning changes
red flag: stuck at the max length: cut off



normal: small and steady (≈ 1e-4 here)
red flag: spikes: an update moved too far



normal: high at first, then settles
red flag: spikes or NaN: unstable

verl's public math baseline: Qwen2.5-Math-7B, a GRPO variant, tested on AIME 2024

Run: https://wandb.ai/verl-org/DAPO%20Reproduction%20on%20verl/runs/ow47vvon (public W&B)
[Yu et al., DAPO: An Open-Source LLM Reinforcement Learning System at Scale 2025]

multisensory intelligence

The Same Step in verl

```
# PPOTrainer.step(), sync mode (verl default)
self._add_batch_to_generate()
batch = self.replay_buffer.sample(batch_size=B)
self.checkpoint_manager.sleep_replicas()
batch = self._balance_batch(batch)
batch = self._compute_old_log_prob(batch)
if self.use_reference_policy:
    batch = self._compute_ref_log_prob(batch)
if self.use_critic:
    batch = self._compute_values(batch)
batch = self._compute_advantage(batch)
batch = self._update_actor(batch)
self.checkpoint_manager.update_weights(self.global_steps)
```

① **rollout**: n responses per prompt

② **reward**: scored as each finishes

free GPU memory for training

give every GPU similar total tokens

log-probs (old, reference) and values

③ **advantage**: the algorithm

④ **update**: one step per mini-batch

⑤ **weight sync** → rollout engine

Code: https://github.com/verl-project/verl/blob/ddf7007/verl/trainer/ppo/v1/trainer_base.py#L555-L611 + trainer_sync.py (simplified)
[Sheng et al., HybridFlow: A Flexible and Efficient RLHF Framework. EuroSys 2025]

multisensory intelligence

Training with verl: What You Write

① tasks → parquet, one row per prompt

```
{"data_source": "openai/gsm8k",
 "prompt": [{"role": "user", "content": q}],
 "reward_model": {"style": "rule",
                  "ground_truth": "72"}}
```

② a reward function

```
def compute_score(data_source, solution_str,
                  ground_truth, extra_info=None, **kw):
    answer = extract_answer(solution_str)
    return float(answer == ground_truth)
```

③ launch (overrides on the config)

```
python3 -m verl.trainer.main_ppo \
  algorithm.adv_estimator=grpo \
  data.train_files=gsm8k/train.parquet \
  data.train_batch_size=1024 \
  actor_rollout_ref.model.path=Qwen/Qwen3-8B \
  actor_rollout_ref.rollout.name=vllm \
  actor_rollout_ref.rollout.n=5 \
  actor_rollout_ref.actor.ppo_mini_batch_size=256 \
  reward.custom_reward_function.path=my_reward.py \
  trainer.n_gpus_per_node=8
```

④ watch the curves
as in the example run

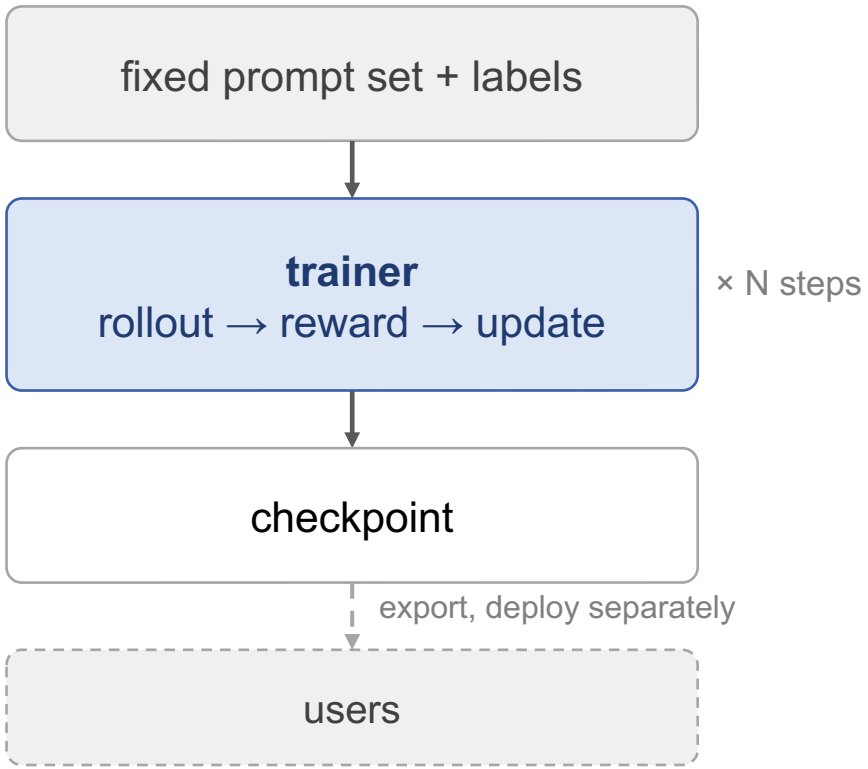
⑤ export: `verl.model_merger merge`
FSDP shards → Hugging Face → serve with vLLM

Docs: https://verl.readthedocs.io/en/latest/start/quickstart.html
Code: https://github.com/verl-project/verl/blob/ddf7007/examples/grpo_trainer/run_qwen3_8b_fsdp.sh

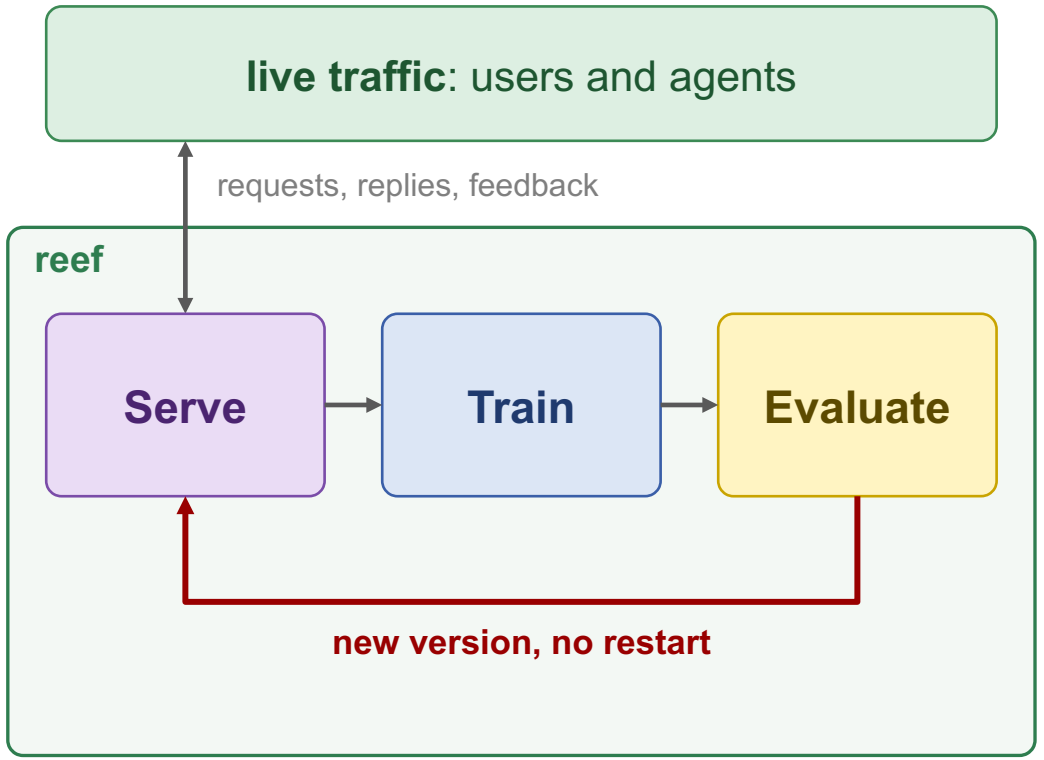
multisensory intelligence

verl vs reef: Dataset vs Live Traffic

verl: train first, deploy later



reef: learn while serving

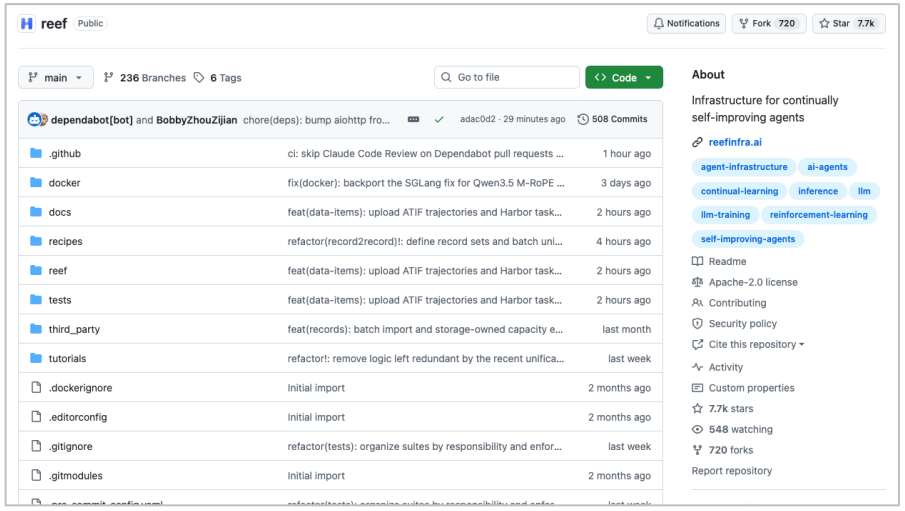


verl is a **trainer** over a dataset you prepared; reef is an **inference server** that learns from its own traffic

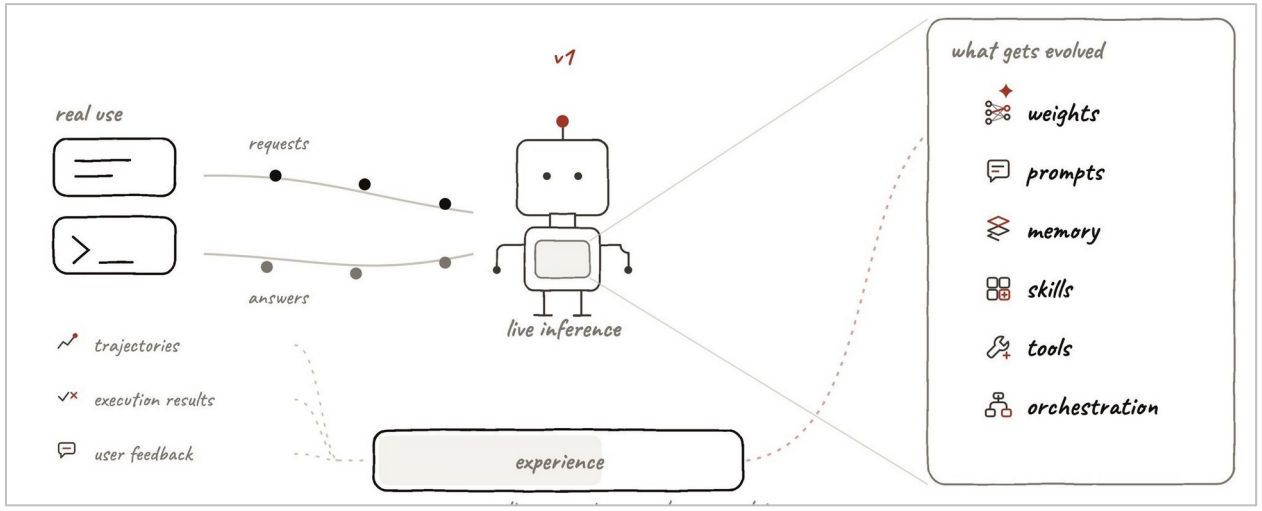
[Sheng et al., HybridFlow: A Flexible and Efficient RLHF Framework. EuroSys 2025] Code: https://github.com/Human-Agent-Society/reef
Blog: https://x.com/ao_qu18465/status/2094867930081337730

multisensory intelligence

reef: Infrastructure for Continual Self-Improving Agents



GitHub
github.com/Human-Agent-Society/reef



Your Inference Server is Secretly a Learner: Open-Sourcing reef for Continual Self-Improving Agents
x.com/ao_qu18465/status/2094867930081337730

multisensory intelligence