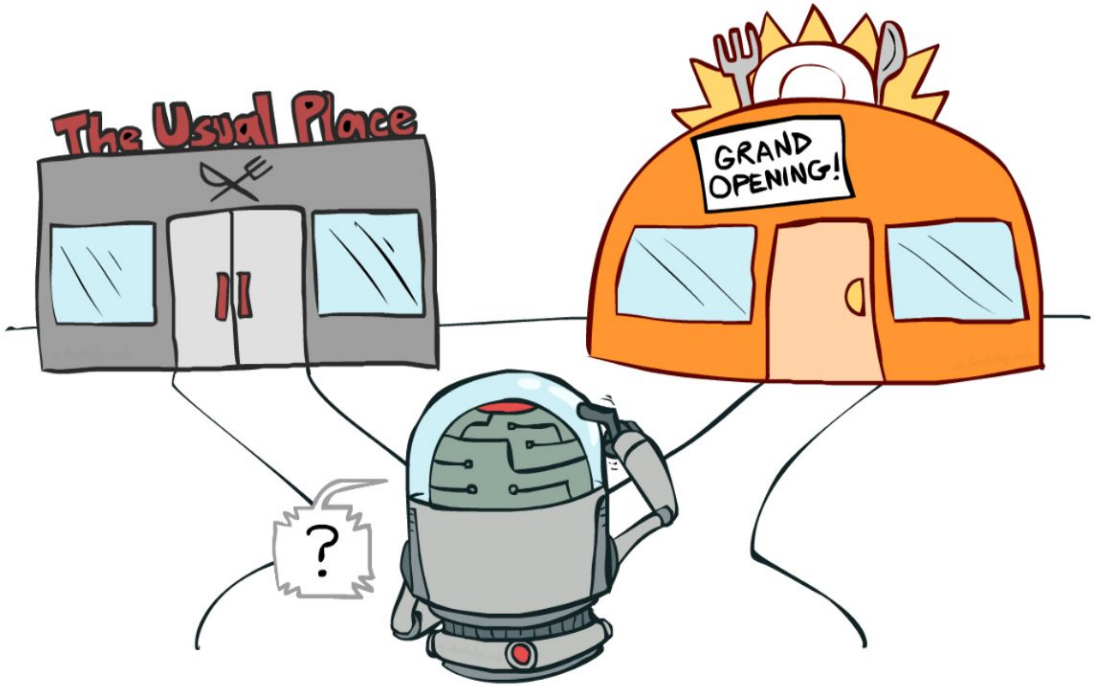
RL overview

2025.07.25
Yifei

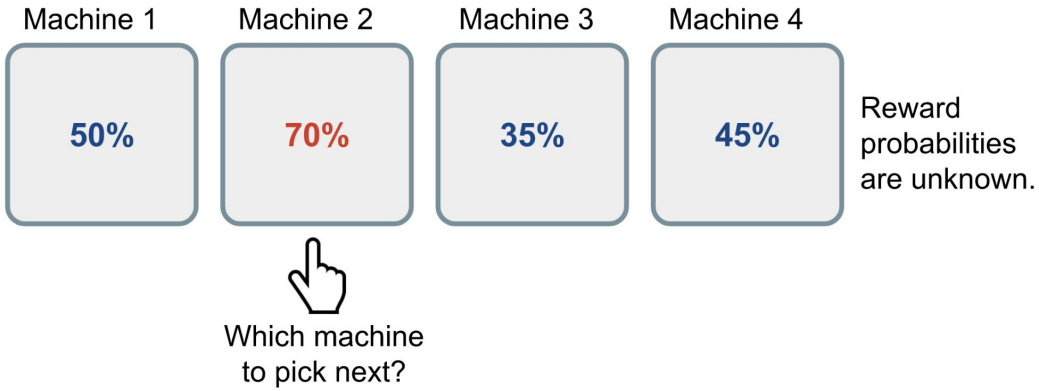
Overview

- Useful concepts in RL
  - Multi-armed bandit
  - Markov Decision Process
  - Value functions
  - Some brief intro to RL algorithm (value-based, policy gradient, actor-critic, etc)
  - Online/offline RL, on-policy/off-policy RL
- RL in LLM
  - PPO
  - RLHF
  - DPO
  - GRPO
  - AReaL (system side optimization for PPO in Reasoning LM)
- Evolutionary algorithm

The known dilemma: Explore VS Exploit



Multi-Armed bandit problem

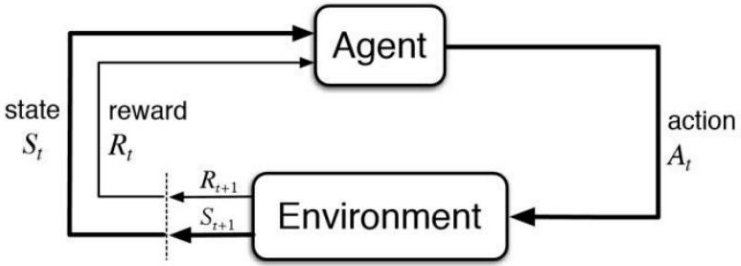


What is the best strategy to maximize the expected total reward over some time period?

Differs from Markov Decision Process because there's no state transition.

What is reinforcement learning?

reinforcement learning



input: $\mathbf{s}_t$ *at each time step*

output: $\mathbf{a}_t$ *at each time step*

data: $(\mathbf{s}_1, \mathbf{a}_1, r_1, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T)$

goal: learn $\pi_\theta : \mathbf{s}_t \rightarrow \mathbf{a}_t$

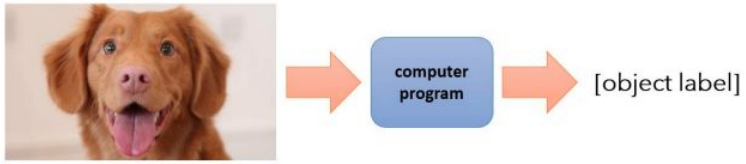
to maximize $\sum_t r_t$

reward

policy

- Data is not i.i.d. Previous output influences future input
- Only know if we succeed or failed (reward)

supervised learning



input: $\mathbf{x}$

output: $\mathbf{y}$

data: $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}$

goal: $f_\theta(\mathbf{x}_i) \approx \mathbf{y}_i$

someone gives this to you

- Normally we assume the data point is i.i.d.
- With ground truth given

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Definitions - Markov Decision Process

Markov decision process $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{T}, r\}$

$\mathcal{S}$ – state space states $s \in \mathcal{S}$ (discrete or continuous)

$\mathcal{A}$ – action space actions $a \in \mathcal{A}$ (discrete or continuous)

$\mathcal{T}$ – transition operator

r – reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$

$r(s_t, a_t)$ – reward

let $\mu_{t,j} = p(s_t = j)$

let $\xi_{t,k} = p(a_t = k)$

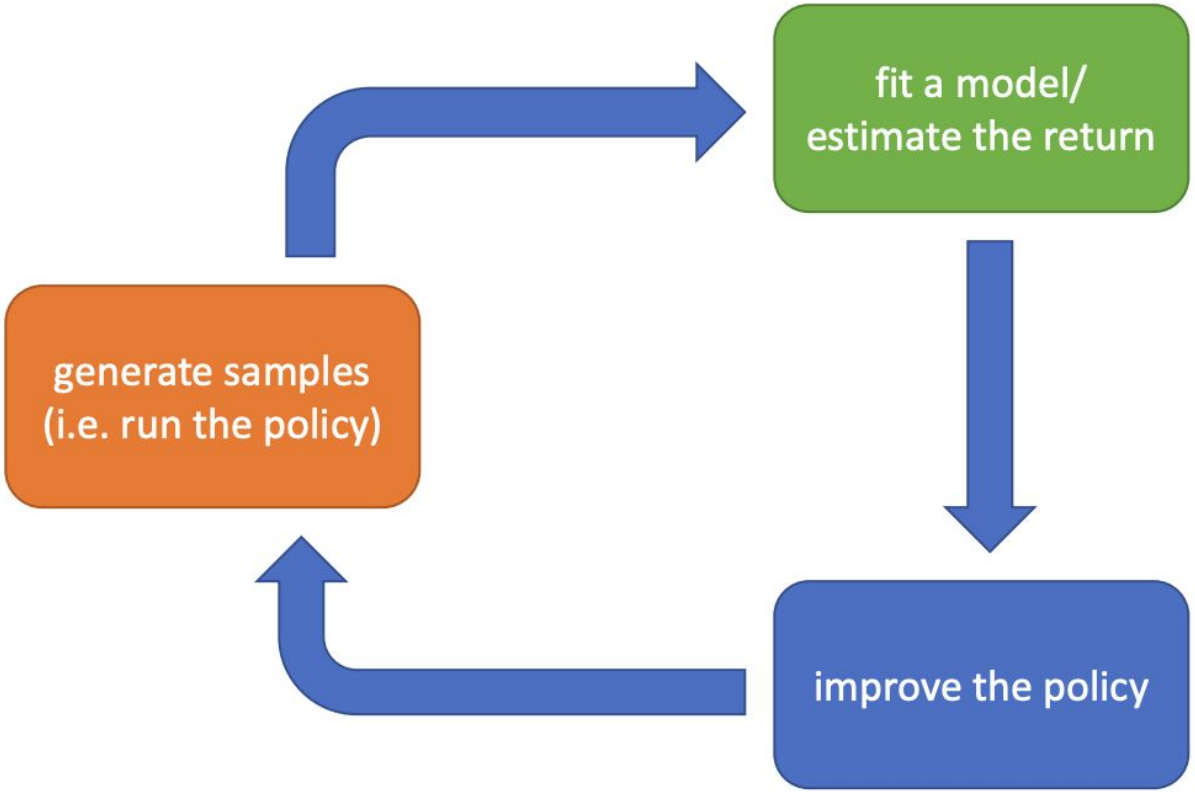
let $\mathcal{T}_{i,j,k} = p(s_{t+1} = i | s_t = j, a_t = k)$

$$\mu_{t+1,i} = \sum_{j,k} \mathcal{T}_{i,j,k} \mu_{t,j} \xi_{t,k}$$

Markov property:
The probability of a certain state and its reward at time t depends on the immediately preceding state and action

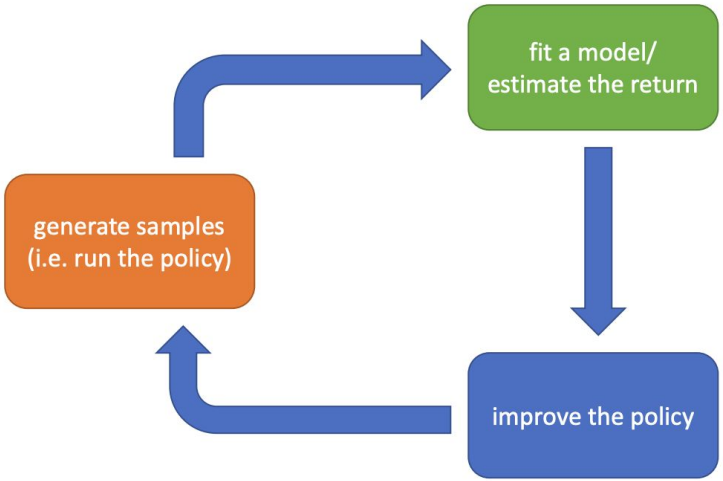
Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Procedure of RL algorithm

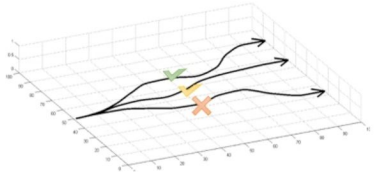


Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Examples



Example 1

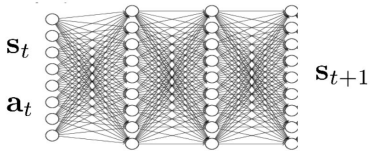


$$J(\theta) = E_\pi \left[\sum_t r_t \right] \approx \frac{1}{N} \sum_{i=1}^{N} \sum_t r_t^i$$

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$$

Example 2

learn f_ϕ such that $\mathbf{s}_{t+1} \approx f_\phi(\mathbf{s}_t, \mathbf{a}_t)$



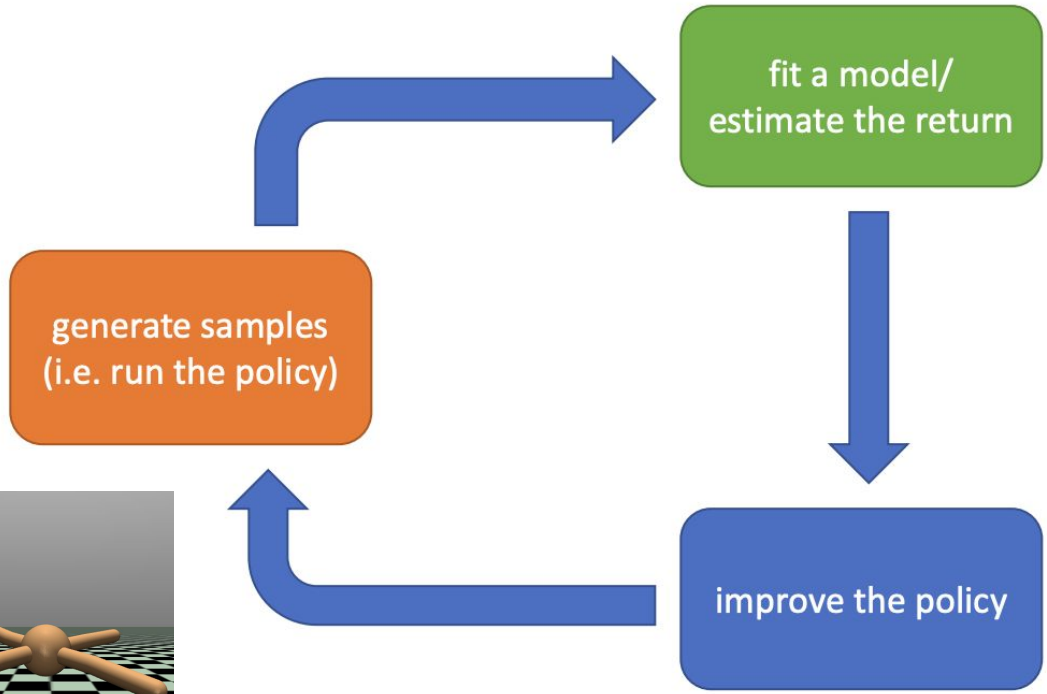
backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Which part(s) can be expensive?

Expensive: use a neural net to learn some state transition function

Cheap: naive formulation like computing the average of reward



Expensive: real robot/car/language response, 1x real time

Cheap: simulation enabled, like chess/MuJoCo

$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

backprop through f_ϕ and r to train $\pi_\theta(\mathbf{s}_t) = \mathbf{a}_t$

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Estimate the return: Value functions

How good it is for an agent to perform a given action at a given state?

State-action function (Q function)

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$$

Total expected rewards from taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$

How good it is for an agent to be in a certain state?

Value function

$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t\right]$$

$$V^{\pi}(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi(\mathbf{a}_t|\mathbf{s}_t)}[Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)]$$

Total expected rewards from state $\mathbf{s}_t$

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

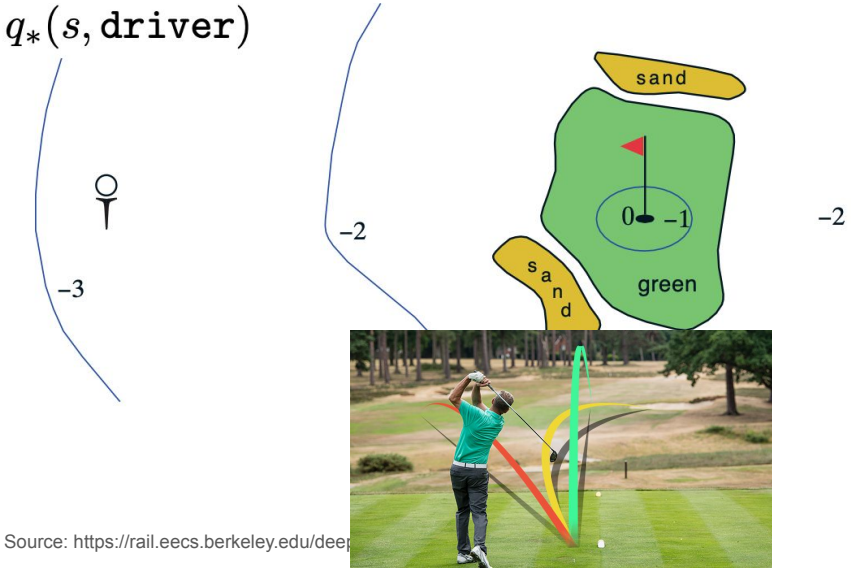
Estimate the return: Value functions

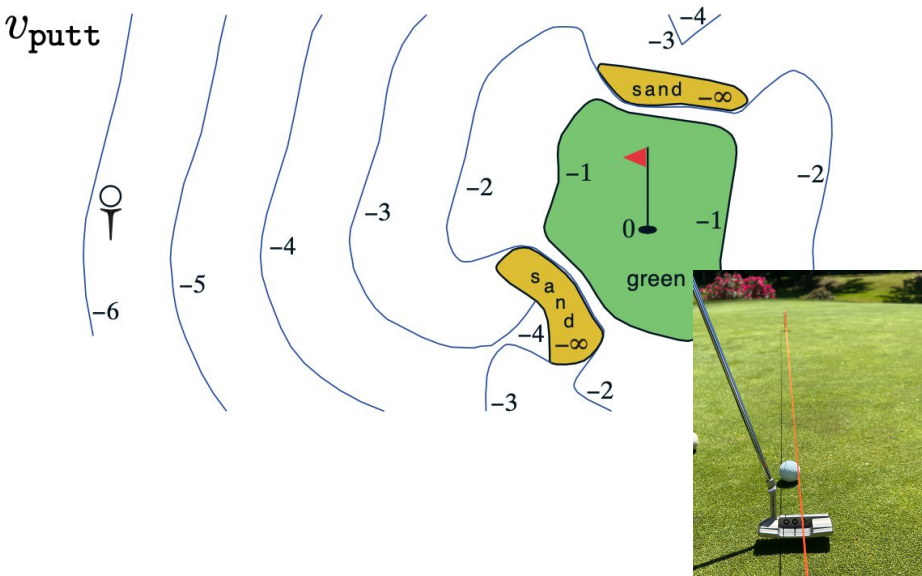
How good it is for an agent to perform a given action at a given state?

$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$$

How good it is for an agent to be in a certain state?

$$V^{\pi}(\mathbf{s}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t\right]$$





Source: https://rail.eecs.berkeley.edu/deep

Types of RL algorithms

Value-based: estimate value function or Q-function of the optimal policy (no explicit policy)

Policy gradients: directly differentiate this objective:

$$\theta^\star = \arg\max_\theta E_{\tau\sim p_\theta(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right]$$

Actor-critic: estimate value function or Q-function of the current policy, use it to improve policy

$p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$

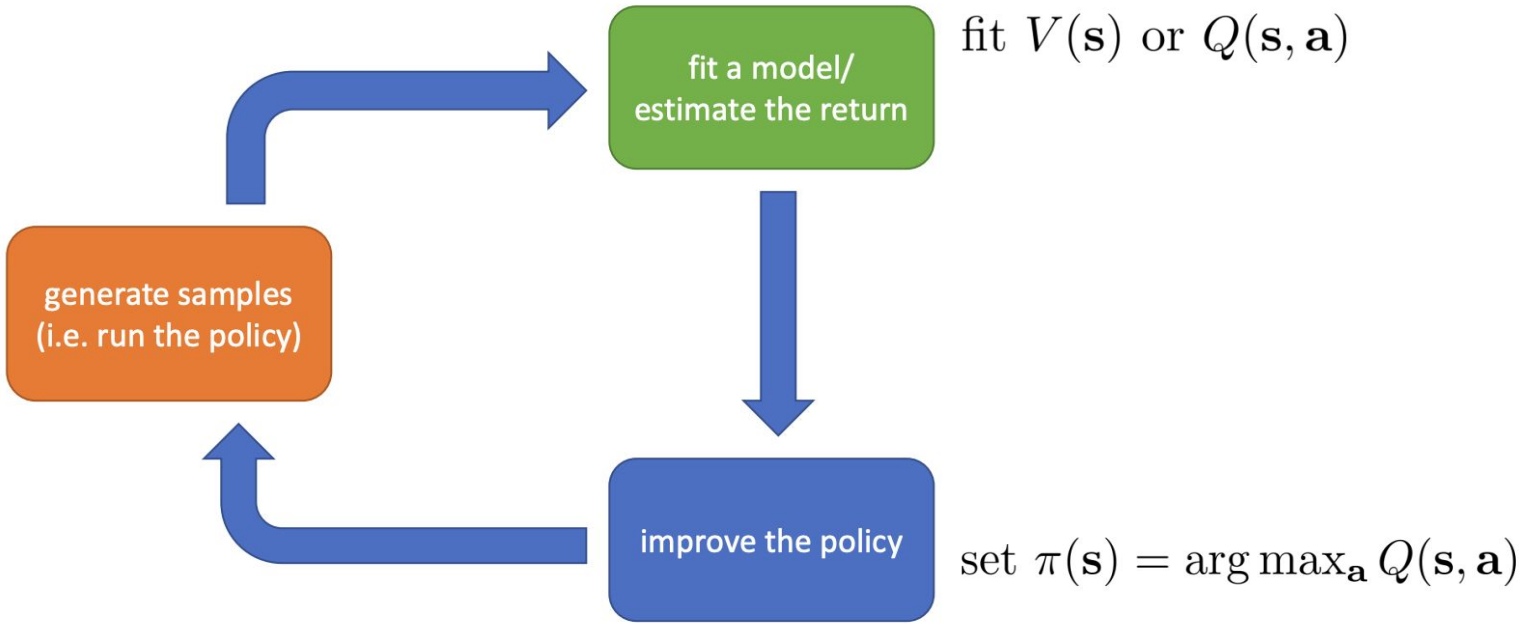
Model-based RL: estimate the transition model (given current state and action, what is the next state)

- Use it for planning (no policy), e.g. optimal control, MCTS
- Use it to improve a policy
- use model to learn value function, e.g. dynamic programming

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Value-based algorithms

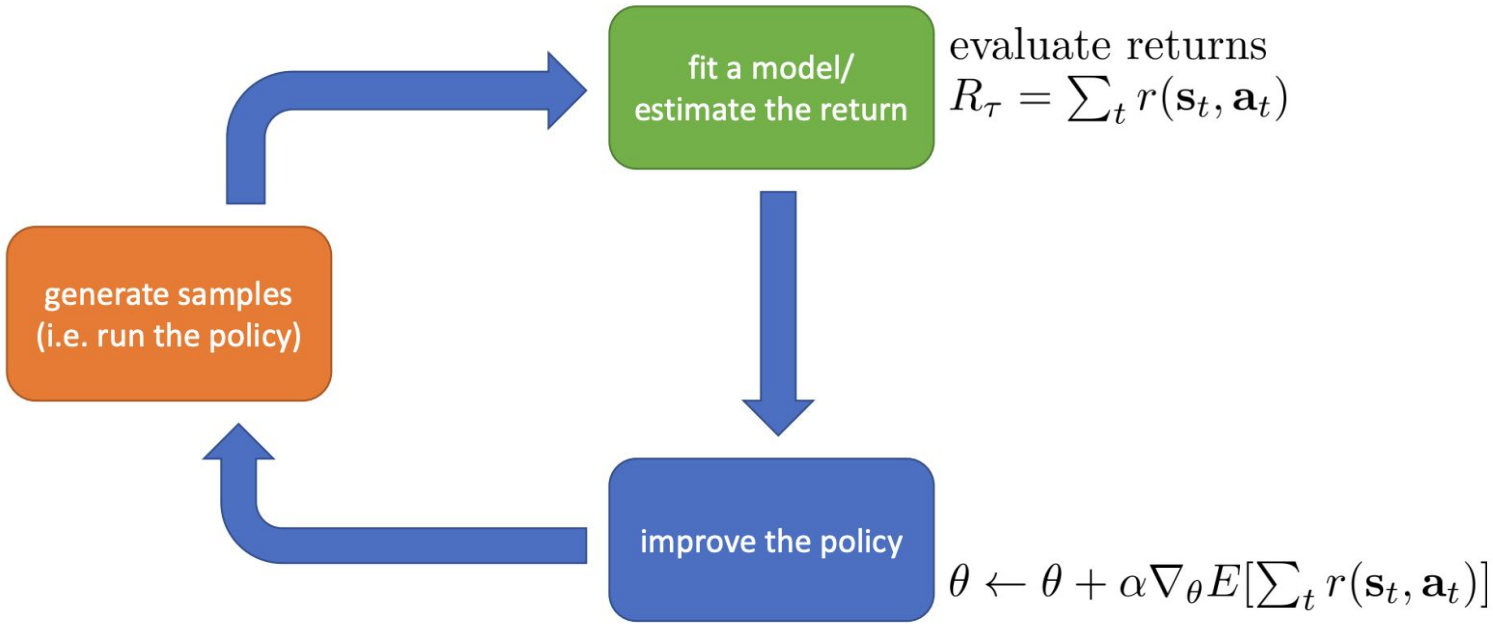
Can we learn the value function or Q function and pick the most promising action?



Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Policy Gradient methods

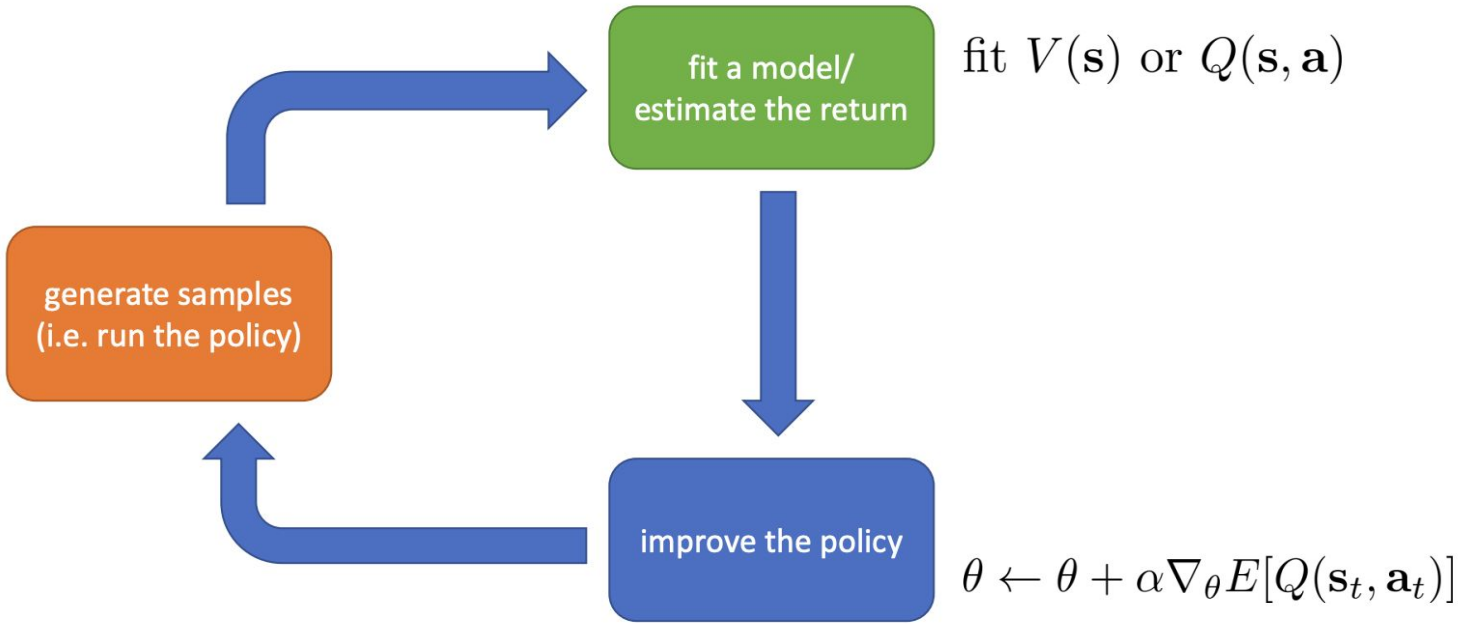
Can we directly learn the policy on how to act?



Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Actor-Critic

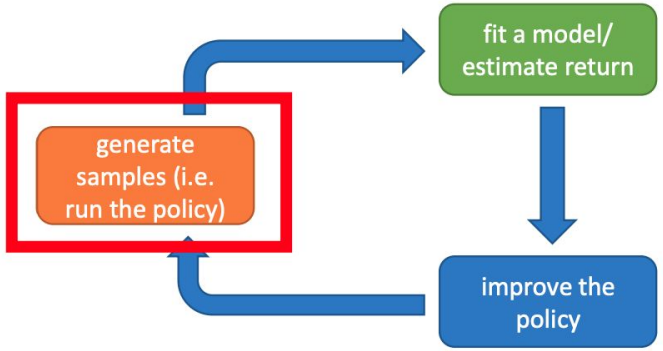
Can we both learn the policy and an estimation of the reward?



Source: https://rail.eecs.berkeley.edu/deeprlcourse/

On-policy RL VS off-policy RL

Sampling can be very time consuming especially for on-policy algorithms.



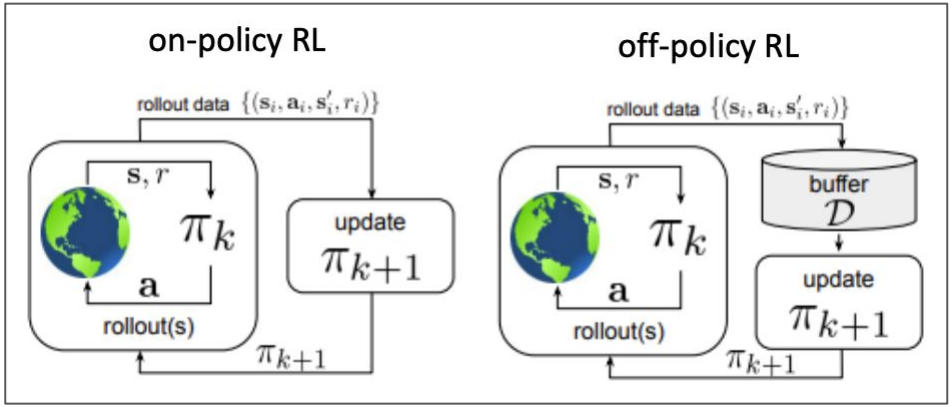
On-policy: each time the policy is changed, even a little bit, we need to generate new samples

Off-policy: able to improve the policy without generating new samples from that policy

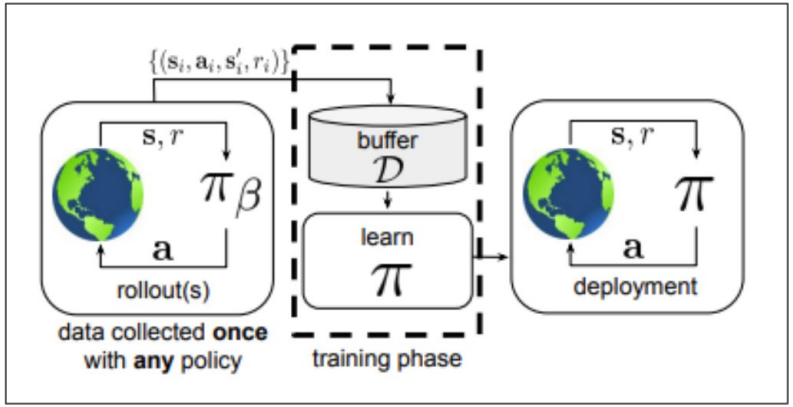
Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Online RL VS Offline RL

Given a lot of data and no interaction with env, can we learn the best policy from it?

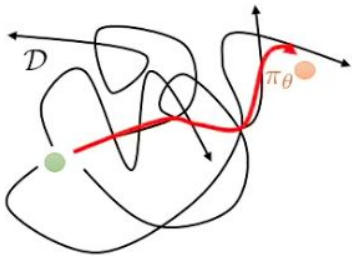


Online RL



Offline RL

It's like 'micro-scale' stitching small good trajectories from a lot of suboptimal trajectories



Summary of RL's terminologies

Environment: everything the agent interacts with

Agent: The learner that makes decision based on its policy

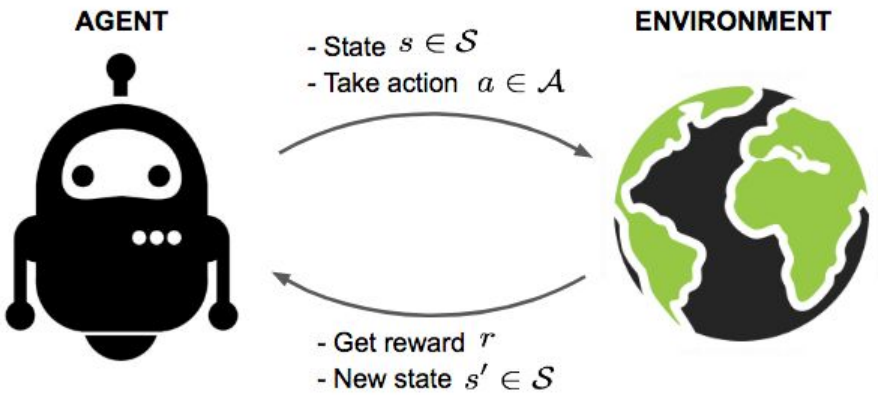
Policy: a strategy to act upon given state

State: encodes current situation

Action: decision made in a given state

Reward: feedback for an action

Value: expected cumulative reward



Goal: By interacting with environment, agent learns a good strategy to maximize future rewards

RL for LLM

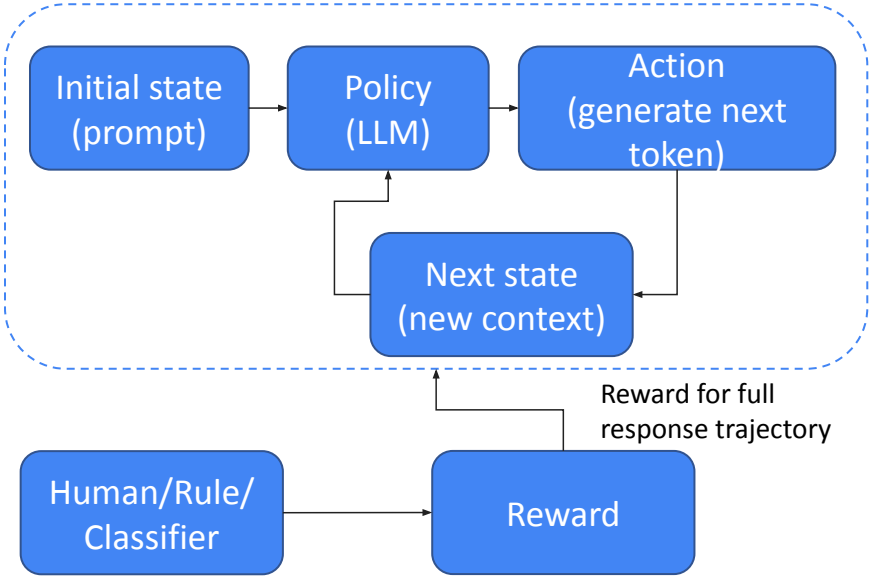
Policy: Language model

State: current context/prompt

Action: next token prediction

Reward: feedback from Human/Rule/Classifier

Trajectory/Rollout: a sequence of tokens generated by LLM



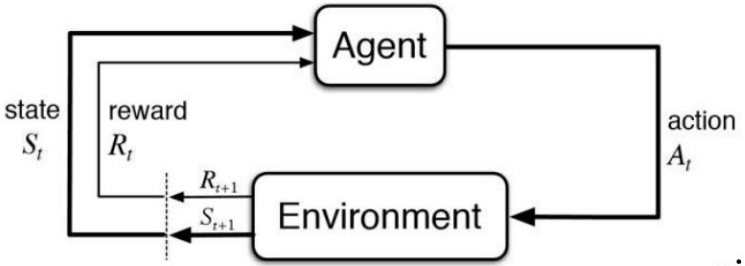
Goal: LLM learns to generate outputs that align with certain reward feedback, can help LLM align with human preference or improve reasoning ability

Proximal Policy Optimization

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Recap

reinforcement learning



pick your own actions

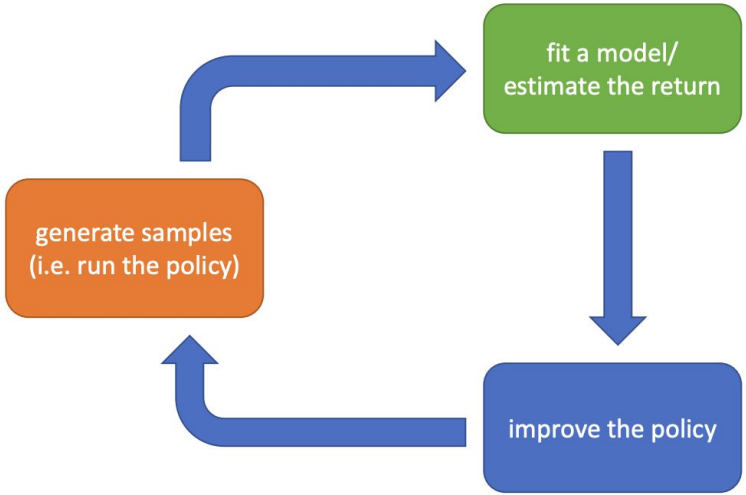
input: $\mathbf{s}_t$ *at each time step*

output: $\mathbf{a}_t$ *at each time step*

data: $(\mathbf{s}_1, \mathbf{a}_1, r_1, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T)$

goal: learn $\pi_\theta : \mathbf{s}_t \rightarrow \mathbf{a}_t$

to maximize $\sum_t r_t$



Policy Gradient methods

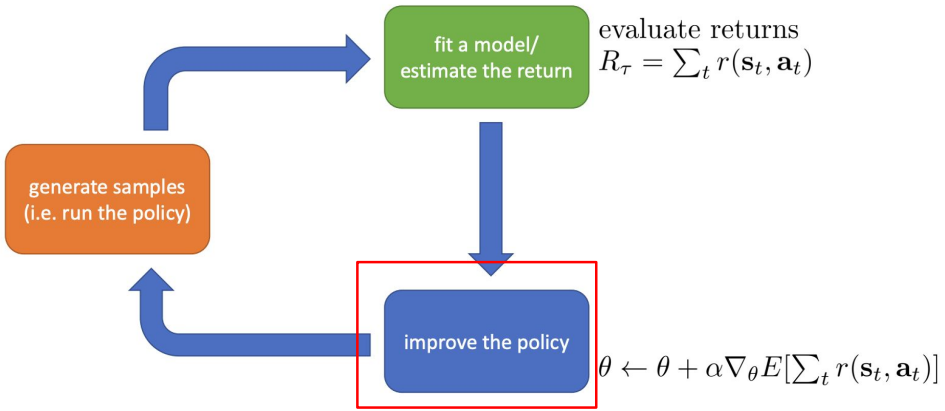
Can we directly learn the policy on how to act?

REINFORCE algorithm:

1. sample $\{\tau^i\}$ from $\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$ (run the policy)
2. $\nabla_\theta J(\theta) \approx \sum_i \left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_t^i|\mathbf{s}_t^i) \left(\sum_{t'=t}^{T} r(\mathbf{s}_{t'}^i, \mathbf{a}_{t'}^i)\right)\right)$
3. $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \hat{Q}_{i,t}^{\pi}$$

"reward to go"



Source: https://rail.eecs.berkeley.edu/deeprlcourse/

Improving the policy gradient

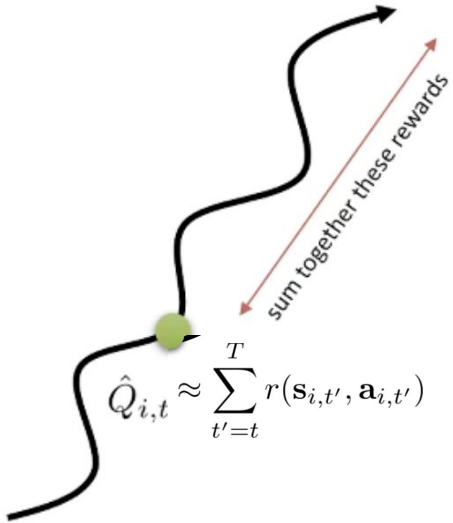
$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t}) \underbrace{\left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'}, \mathbf{a}_{i,t'}) \right)}$$

"reward to go"

$$\hat{Q}_{i,t}$$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

can we get a better estimate?



Improving the policy gradient

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})\underbrace{\left(\sum_{t'=1}^{T} r(\mathbf{s}_{i,t'},\mathbf{a}_{i,t'})\right)}_{\text{“reward to go”}}$$

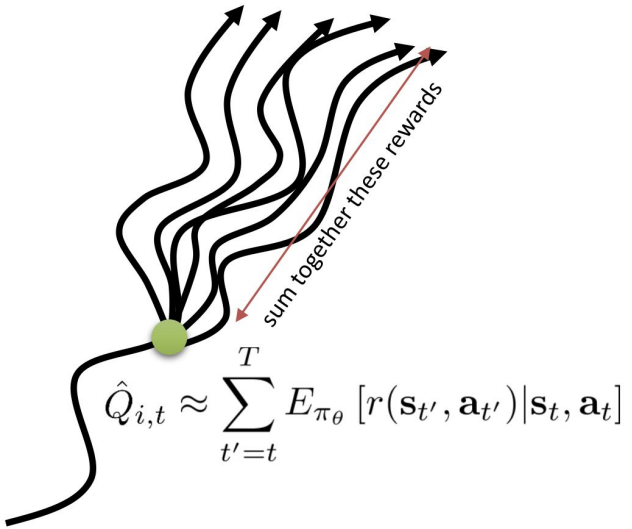
$\hat{Q}_{i,t}$

$\hat{Q}_{i,t}$: estimate of expected reward if we take action $\mathbf{a}_{i,t}$ in state $\mathbf{s}_{i,t}$

can we get a better estimate?

$Q(\mathbf{s}_t,\mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'},\mathbf{a}_{t'})|\mathbf{s}_t,\mathbf{a}_t\right]$: true *expected* reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t}|\mathbf{s}_{i,t})Q(\mathbf{s}_{i,t},\mathbf{a}_{i,t})$$



Can we have a baseline of expected reward?

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$: true *expected* reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) (Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - b)$$

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

We can take the average Q value as baseline

A better baseline can be a function that depends on state

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t)]$$

Can we have a baseline of expected reward?

$Q(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$: true *expected* reward-to-go

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(\mathbf{a}_{i,t} | \mathbf{s}_{i,t}) (Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - b)$$

$Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t}) - V(\mathbf{s}_{i,t})$

This is called the Advantage function

$$b_t = \frac{1}{N} \sum_i Q(\mathbf{s}_{i,t}, \mathbf{a}_{i,t})$$

We can take the average Q value as baseline

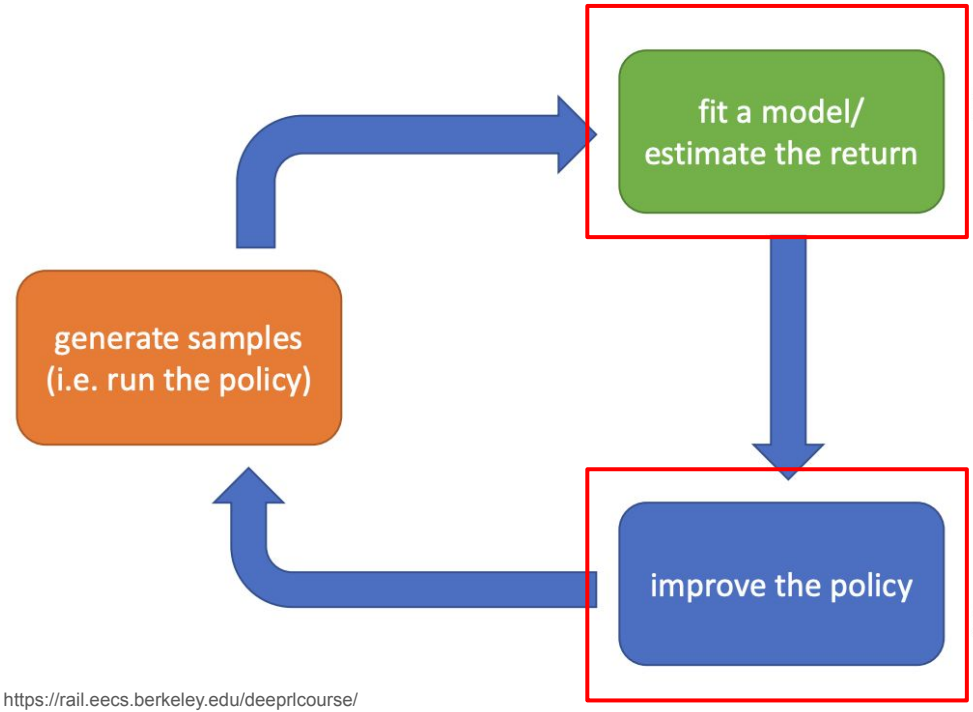
A better baseline can be a function that depends on state

$$V(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)}[Q(\mathbf{s}_t, \mathbf{a}_t)]$$

Intuition:
We increase the probability of actions that is better than average and decrease the probability of actions that is worse than average

Actor-Critic

Can we both learn the policy and an estimation of the reward?



fit Q^π, V^π, or A^π

$Q^\pi(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]$: total reward from taking $\mathbf{a}_t$ in $\mathbf{s}_t$

$V^\pi(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^\pi(\mathbf{s}_t, \mathbf{a}_t)]$: total reward from $\mathbf{s}_t$

$A^\pi(\mathbf{s}_t, \mathbf{a}_t) = Q^\pi(\mathbf{s}_t, \mathbf{a}_t) - V^\pi(\mathbf{s}_t)$: how much better $\mathbf{a}_t$ is

$$\theta \leftarrow \theta + \alpha \nabla_\theta E[Q(\mathbf{s}_t, \mathbf{a}_t)]$$

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

State and state-action functions

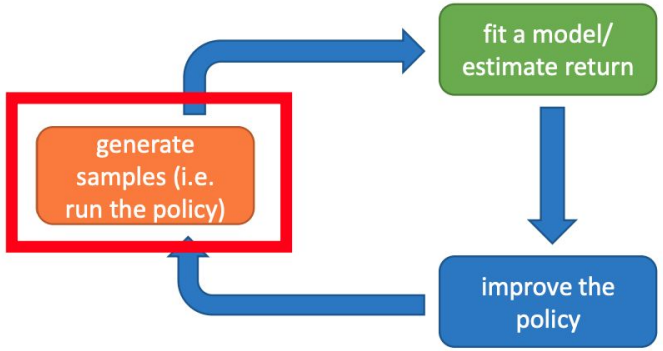
$$Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(\mathbf{s}_{t'}, \mathbf{a}_{t'}) | \mathbf{s}_t, \mathbf{a}_t\right]: \text{ total reward from taking } \mathbf{a}_t \text{ in } \mathbf{s}_t$$

$$V^{\pi}(\mathbf{s}_t) = E_{\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}[Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t)]: \text{ total reward from } \mathbf{s}_t$$

$$A^{\pi}(\mathbf{s}_t, \mathbf{a}_t) = Q^{\pi}(\mathbf{s}_t, \mathbf{a}_t) - V^{\pi}(\mathbf{s}_t): \text{ how much better } \mathbf{a}_t \text{ is}$$

On-policy RL VS off-policy RL

Sampling can be very time consuming especially for on-policy algorithms.



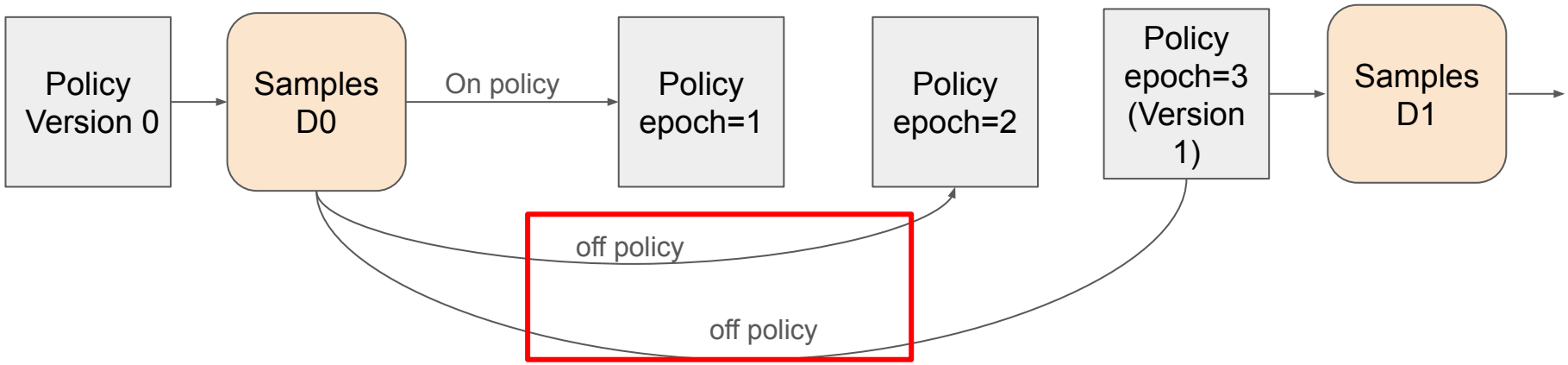
On-policy: each time the policy is changed, even a little bit, we need to generate new samples

Off-policy: able to improve the policy without strictly using the samples generated from that policy

Source: https://rail.eecs.berkeley.edu/deeprlcourse/

An illustration

Assume we set epoch_num=3 and batch_size=entire_train_set



Could lead to large policy updates -> need constraints!

Problem? Data inefficiency!

- On-policy method: for each new policy, we need to generate completely new trajectories
- The trajectory is thrown out after just one gradient update
- Performing multiple steps of optimization on the loss with the same trajectory is appealing but can lead to destructively large policy updates

Can we add a constraint on the updated policy so that it does not diverge too much from the policy that generated the trajectory?

Policy Gradient methods

Estimator of **Advantage**: how much better (or worse) an action a_t at is compared to the policy's average expected value at state s_t

Objective for Policy Gradient methods

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

Empirical average over a finite batch of samples

Commonly used gradient estimator

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \log \pi_\theta(a_t \mid s_t) \hat{A}_t \right]$$

Trust Region Policy Optimization (TRPO)

Importance ratio from the importance sampling

Hard constraint version:

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right]$$

$$\text{subject to} \quad \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \leq \delta$$

Penalty version:

$$\underset{\theta}{\text{maximize}} \, \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

TRPO uses a **hard constraint** rather than a penalty because it is hard to choose a single value of β that performs well across different problems—or even within a single problem, where the characteristics change over the course of learning

Proximal Policy Optimization

Motivation:

- The hard constraint helps in the training process from TRPO.
- Maybe the constraint is not a strict constraint: Does it matter if we only break the constraint just a few times?
- What if we treat it as a “soft” constraint?
  - Add *proximal value* to objective function

1. Adaptive KL penalty
2. Clipped Surrogate

PPO with adaptive KL Penalty

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t - \beta \, \text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)] \right]$$

Hard to pick β value → use adaptive β

$$\text{Compute } d = \hat{\mathbb{E}}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]]$$

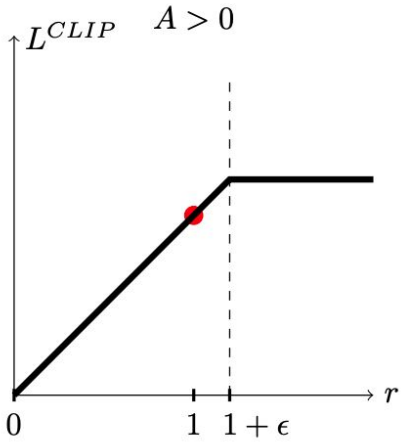
- If $d < d_{\text{targ}}/1.5$, $\beta \leftarrow \beta/2$
- If $d > d_{\text{targ}} \times 1.5$, $\beta \leftarrow \beta \times 2$

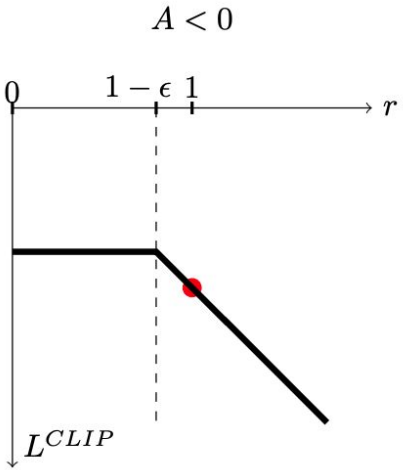
- 1.5 and 2 are heuristically chosen, model are not sensitive to them
- This version perform worse than the clipped surrogate objective

PPO with Clipped Objective

$$\underset{\theta}{\text{maximize}} \quad \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \hat{A}_t \right] \qquad r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \qquad r(\theta_{\text{old}}) = 1$$

Fluctuation happens when r changes too much -> limit r within a range?





$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right]$$

PPO with Clipped Objective

Algorithm 5 PPO with Clipped Objective

Input: initial policy parameters θ_0, clipping threshold ϵ

for $k = 0, 1, 2, \ldots$ **do**

Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$

Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \cdots + \cdots + (\gamma\lambda)^{T-t+1}\delta_{T-1},$$
$$\text{where} \quad \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$$

Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}^{CLIP}(\theta)$$

by taking K steps of minibatch SGD (via Adam), where

$$\mathcal{L}_{\theta_k}^{CLIP}(\theta) = \mathop{\mathrm{E}}_{\tau \sim \pi_k}\left[\sum_{t=0}^{T}\left[\min(r_t(\theta)\hat{A}_t^{\pi_k}, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t^{\pi_k})\right]\right]$$

end for

algorithm	avg. normalized score
No clipping or penalty	-0.39
Clipping, $\epsilon = 0.1$	0.76
Clipping, $\epsilon = 0.2$	**0.82**
Clipping, $\epsilon = 0.3$	0.70
Adaptive KL $d_{\text{targ}} = 0.003$	0.68
Adaptive KL $d_{\text{targ}} = 0.01$	0.74
Adaptive KL $d_{\text{targ}} = 0.03$	0.71
Fixed KL, $\beta = 0.3$	0.62
Fixed KL, $\beta = 1.$	0.71
Fixed KL, $\beta = 3.$	0.72
Fixed KL, $\beta = 10.$	0.69

Table 1: Results from continuous control benchmark. Average normalized scores (over 21 runs of the algorithm, on 7 environments) for each algorithm / hyperparameter setting . β was initialized at 1.

Reinforcement Learning with Human Feedback

RLHF workflow

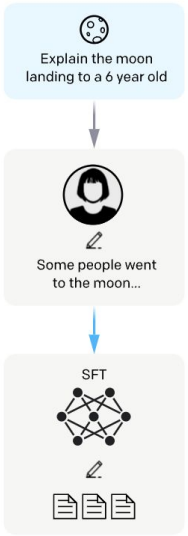
Step 1 **SFT**

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



Step 2 **Reward Modeling**

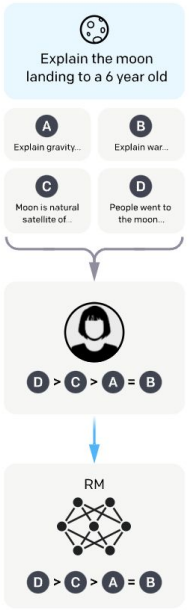
Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.

LM + linear layer to output scalar



Step 3 **RL finetuning**

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

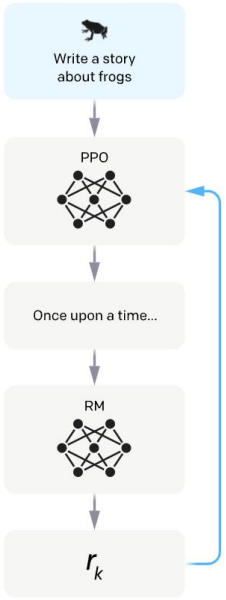
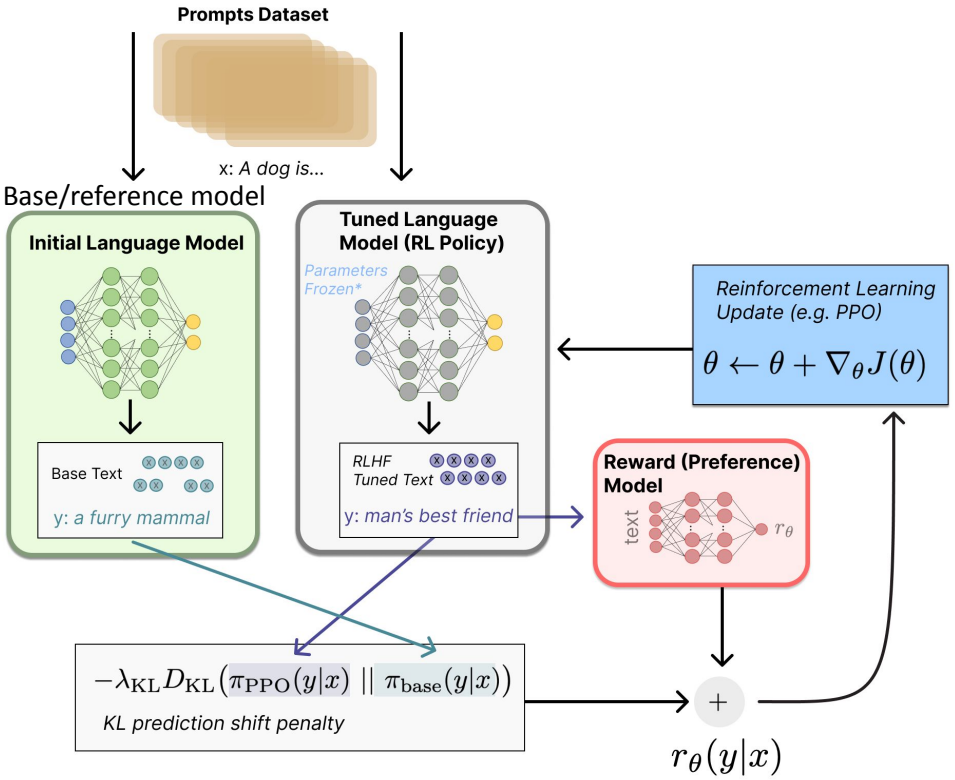


Image source: https://arxiv.org/pdf/2203.02155

PPO in RLHF

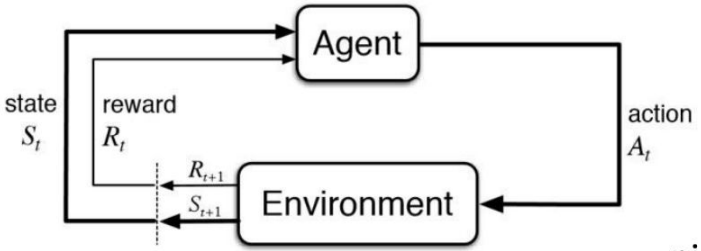
- *Base model*: frozen, used to compute KL divergence
- *Policy model (actor model):* the LM we want to finetune
- *Reward model*: generate an outcome reward for rollout



Direct Preference Optimization

Recap: Reinforcement Learning

reinforcement learning



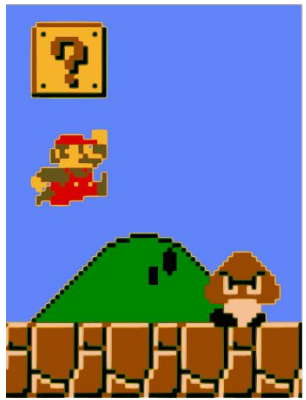
input: $\mathbf{s}_t$ *at each time step*

output: $\mathbf{a}_t$ *at each time step*

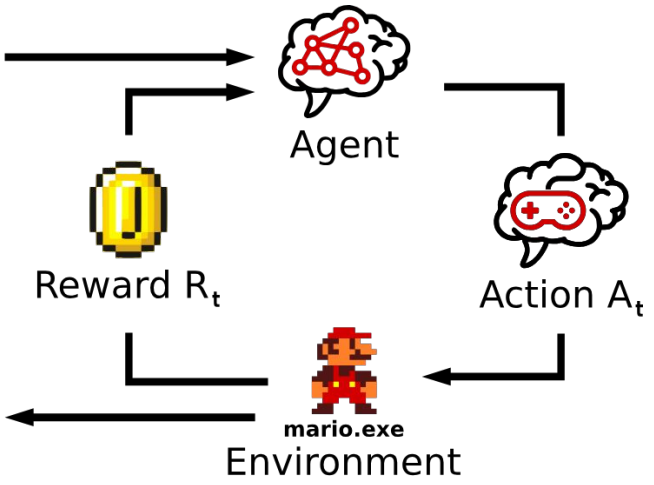
data: $(\mathbf{s}_1, \mathbf{a}_1, r_1, \ldots, \mathbf{s}_T, \mathbf{a}_T, r_T)$

goal: learn $\pi_\theta : \mathbf{s}_t \rightarrow \mathbf{a}_t$
to maximize $\sum_t r_t$

pick your
own actions



States s_t



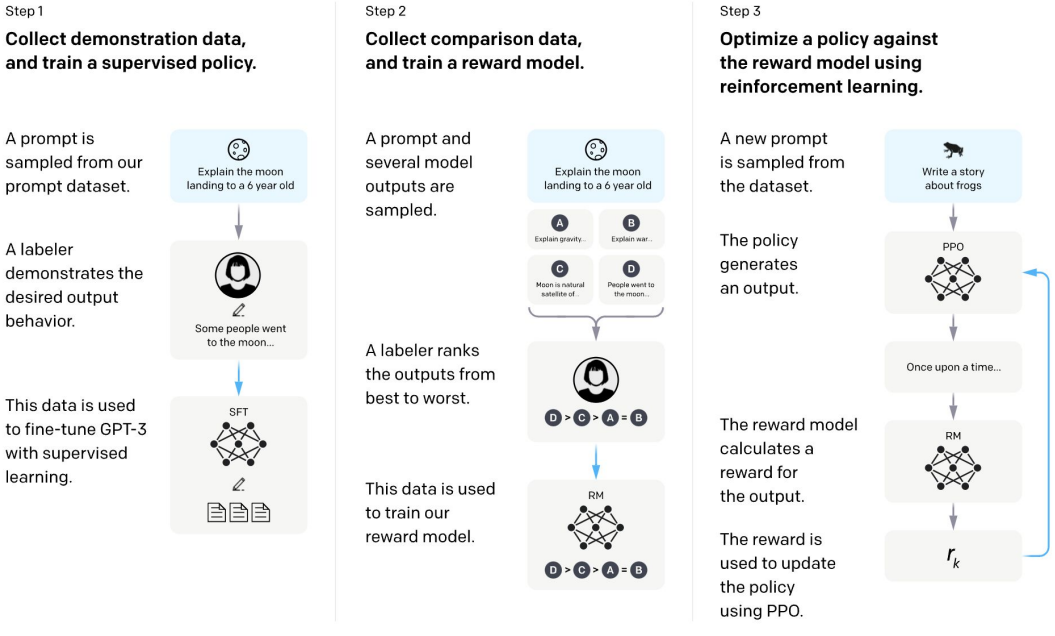
Recap: PPO and RLHF

PPO Objective functions:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right]$$

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}\hat{A}_t - \beta\,\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]\right]$$

RLHF workflow:

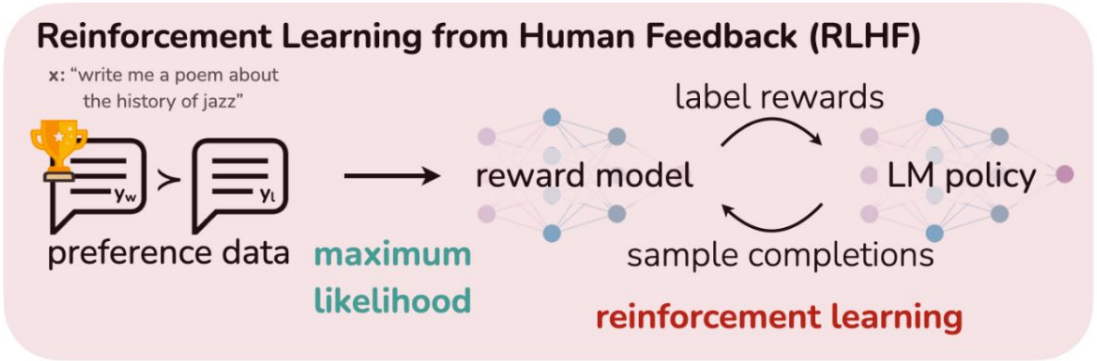


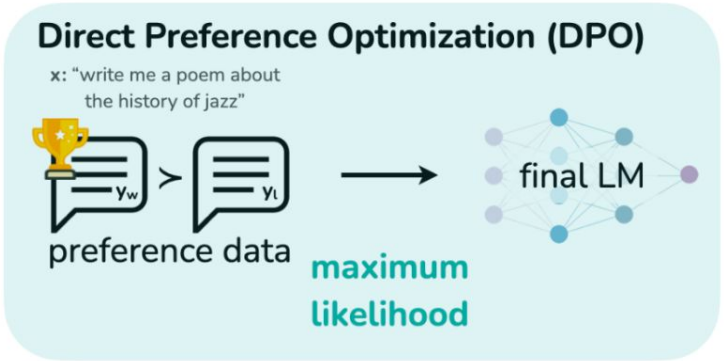
Motivation

Building and training a reward model can be complicated

How to simplify the process of RLHF?

Method - Main idea





Directly learning from preference data
No need for explicitly fitting a reward model!

RLHF workflow

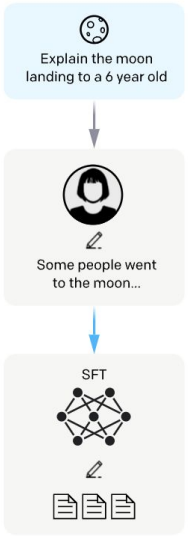
Step 1 **SFT**

Collect demonstration data, and train a supervised policy.

A prompt is sampled from our prompt dataset.

A labeler demonstrates the desired output behavior.

This data is used to fine-tune GPT-3 with supervised learning.



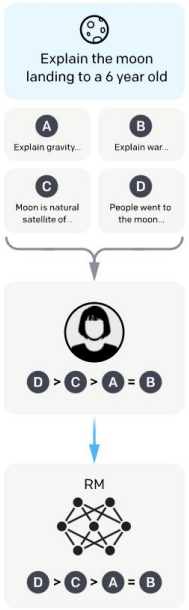
Step 2 **Reward Modeling**

Collect comparison data, and train a reward model.

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

This data is used to train our reward model.



Step 3 **RL finetuning**

Optimize a policy against the reward model using reinforcement learning.

A new prompt is sampled from the dataset.

The policy generates an output.

The reward model calculates a reward for the output.

The reward is used to update the policy using PPO.

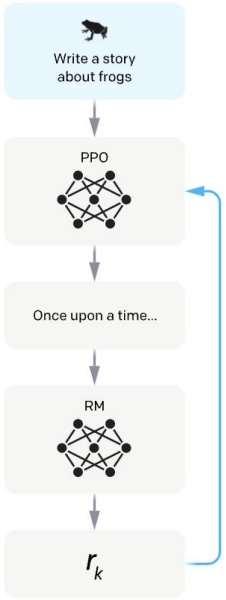


Image source: https://arxiv.org/pdf/2203.02155

Reward modeling phase in RLHF

Feedback comes as **preferences over model samples:** $\mathcal{D} = \{x^i, y_w^i, y_l^i\}$

Prompt

Preferred response

Dispreferred response

Bradley-Terry Model connects rewards to preferences:

Reward assigned to **preferred** and **dispreferred** responses

$$p(y_w \succ y_l \mid x) = \sigma(r(x, y_w) - r(x, y_l))$$

Train the reward model by **minimizing negative log likelihood:**

$$\mathcal{L}_R(\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

Method - Recap on RLHF

RL fine-tuning phase: Learn a policy that **maximize reward** while **stay close** to the reference policy

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\text{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\text{ref}}(y \mid x) \right]$$

PPO objective used in RLHF paper

Method - Recap on RLHF

RL fine-tuning phase: Learn a policy that **maximize reward** while **stay close** to the reference policy

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y) \right] - \beta \mathbb{D}_{\mathrm{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x) \right]$$

Reward from reward model

Policy learnt from PPO

Policy right after SFT

Method - Recap on RLHF

RL fine-tuning phase: Learn a policy that **maximize reward** while **stay close** to the reference policy

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\mathbf{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\text{ref}}(y \mid x)\right]$$

Maximizes the rewards

Prevents the learnt policy from changing too much from the SFT model

Direct Preference Optimization

RLHF Objective

(get **high reward**, stay **close** to reference model)

any reward function

$$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi(y|x)} \left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}(\pi(\cdot \mid x)\|\pi_{\mathrm{ref}}(\cdot \mid x))$$

Closed-form Optimal Policy

(write **optimal policy** as function of **reward function**; from prior work)

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

$$\sum_y \pi_{\mathrm{ref}}(y \mid x)\exp\left(\tfrac{1}{\beta}r(x,y)\right)$$

intractable!

Screenshot source: https://web.stanford.edu/class/cs234/CS234Spr2024/slides/dpo_slides.pdf

Direct Preference Optimization

RLHF Objective

(get **high reward**, stay **close** to reference model)

any reward function

$$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi(y|x)} \left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}(\pi(\cdot \mid x)\|\pi_{\mathrm{ref}}(\cdot \mid x))$$

Closed-form Optimal Policy

(write **optimal policy** as function of **reward function**; from prior work)

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

Rearrange

(write **any reward function** as function of **optimal policy**)

$$r(x,y) = \underbrace{\beta\log\frac{\pi^*(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta\log Z(x)}_{\text{some parameterization of a reward function}}$$

Screenshot source: https://web.stanford.edu/class/cs234/CS234Spr2024/slides/dpo_slides.pdf

Direct Preference Optimization

A loss function on reward functions

+

A transformation between reward functions and policies

=

A loss function on policies

Derived from the Bradley-Terry model of human preferences:

$$\mathcal{L}_R(r, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma(r(x, y_w) - r(x, y_l))\right]$$

$$r_{\pi_\theta}(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

Reward of **preferred** response

Reward of **dispreferred** response

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

Screenshot source: https://web.stanford.edu/class/cs234/CS234Spr2024/slides/dpo_slides.pdf

Takeaways

- DPO removes the RL training loop from RLHF

- DPO is simpler and computationally cheaper than PPO

- DPO is optimizing the same objective as RLHF

Is RLHF really RL?

Contextual Bandit?

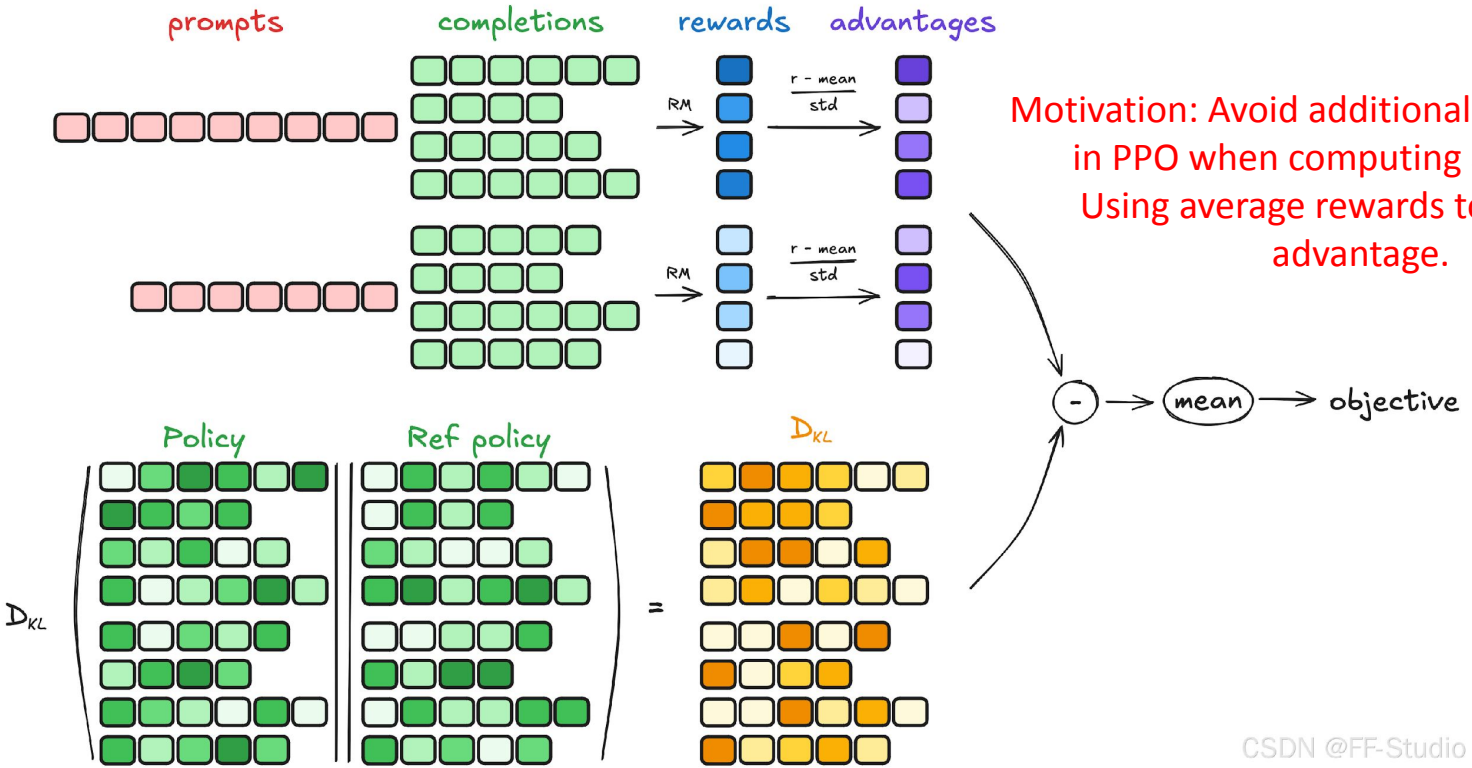
- Reward is only given after the sequence is completely generated
- LM itself can be regarded as a direct mapping from inputs to distributions over completions, rather than a sequential decision-making agent in the space of tokens

Offline RL / Supervised learning?

- The environment consists the policy(LLM) itself
- No new info from the environment for LLM except for the preference data

Reference: https://iclr-blogposts.github.io/2024/blog/rlhf-without-rl/

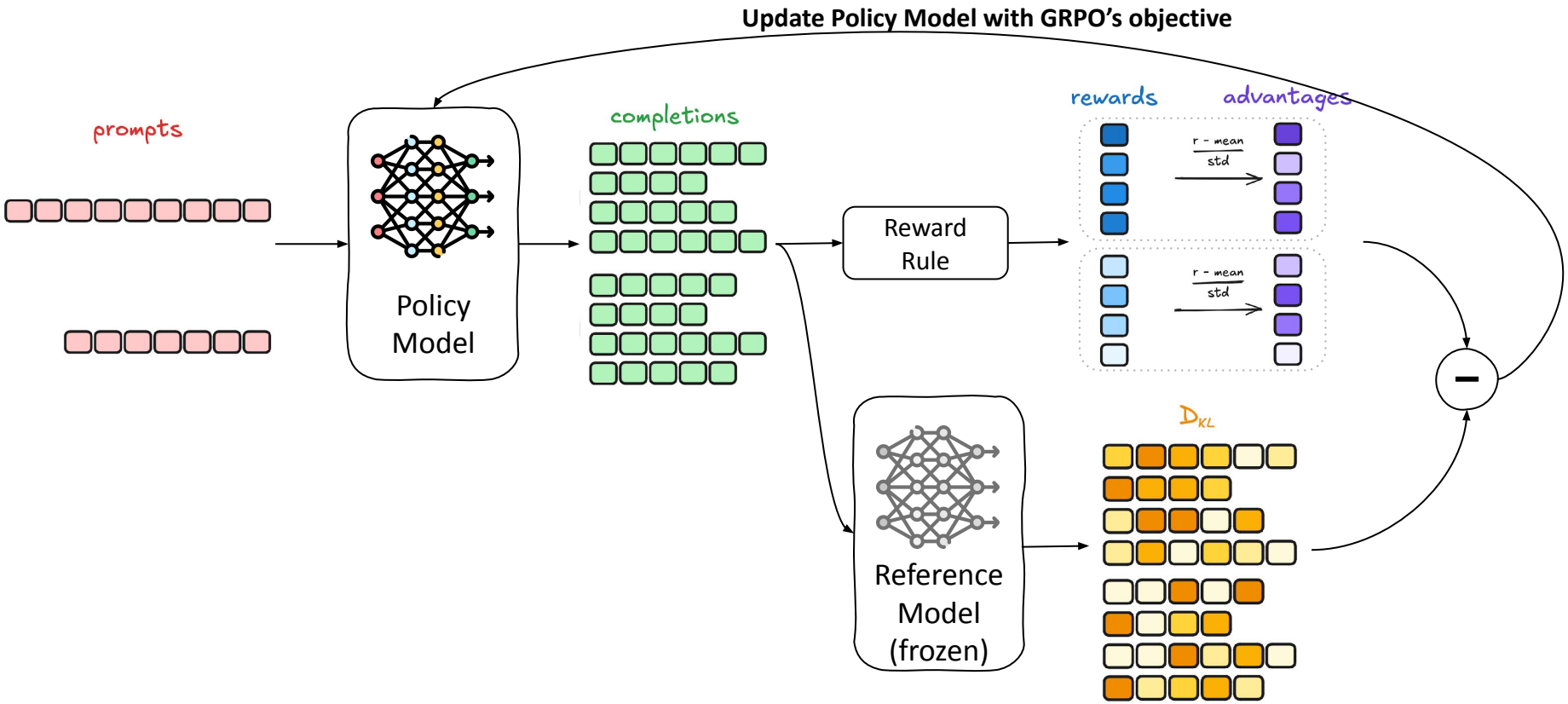
Group Relative Policy Optimization (GRPO)



Motivation: Avoid additional value function in PPO when computing advantage. Using average rewards to compute advantage.

Image source: https://jishuzhan.net/article/1887744102317953026

GRPO Workflow



GRPO – An example

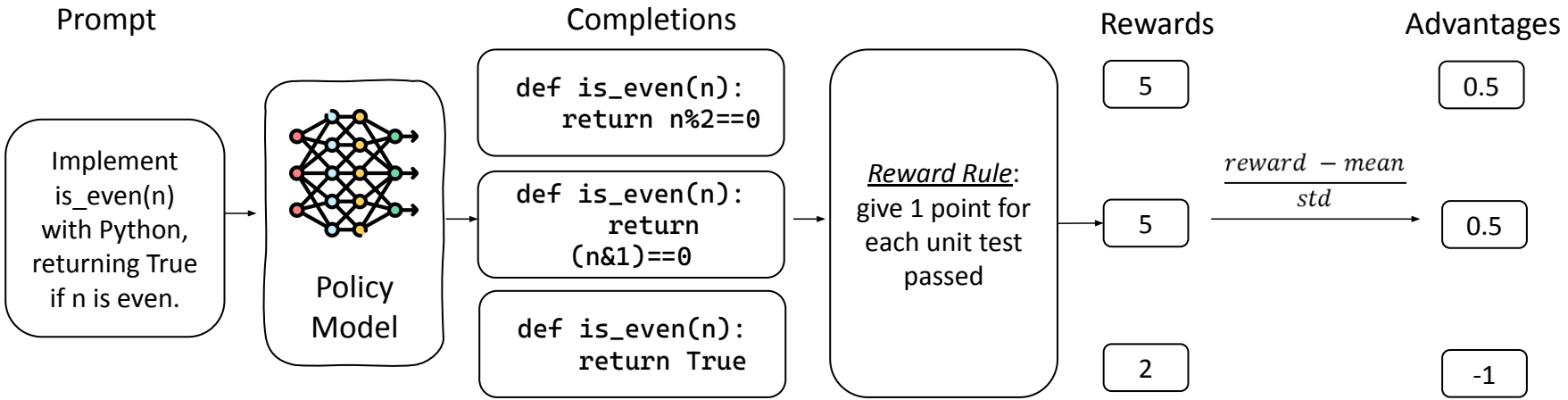


Image source: https://jishuzhan.net/article/1887744102317953026

Dive a little bit into the RL infra side…

AReaL: A Large-Scale Asynchronous Reinforcement Learning System for Language Reasoning

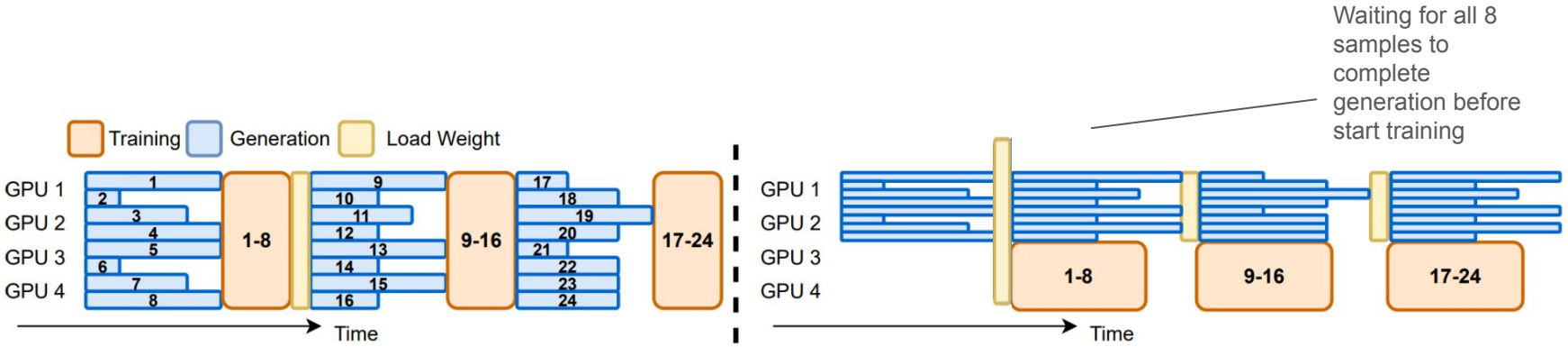
Wei Fu[12], Jiaxuan Gao[12], Xujie Shen[2], Chen Zhu[2], Zhiyu Mei[12], Chuyi He[2], Shusheng Xu[12], Guo Wei[2], Jun Mei[2], Jiashu Wang[23], Tongkai Yang[2], Binhang Yuan[3], Yi Wu[12]

[1] IIIS, Tsinghua University, [2] Ant Research, [3] HKUST
fuwth17@gmail.com, jxwuyi@gmail.com

Motivation to improve RL training system for reasoning LM

RL can be used to improve reasoning abilities in LLMs

- On-policy algorithms require rollouts to be generated by the latest policy
- Naive synchronous pipeline
  - alternates the generation phase and training phase strictly
  - all nodes wait for the slowest task in the batch to finish before training can start
  - chain-of-thought lengths vary greatly, creating a “longest task bottleneck”
  - leads to low GPU utilization as many GPUs sit idle waiting for others to finish generation.



AReaL's async generation

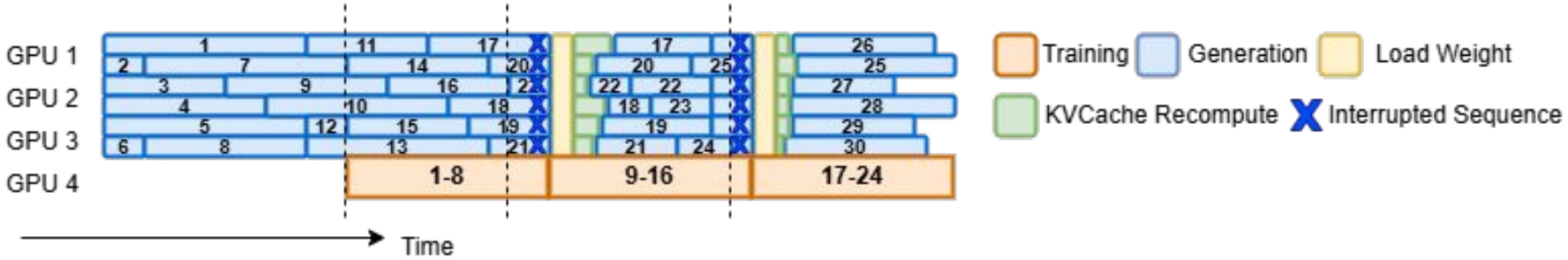


Figure 3: Illustration of generation management in AReaL. Vertical lines show the ready time for the next step training. Blue crosses show the interrupted requests when new parameters arrive.

Data staleness → staleness control during generation task submission

Inconsistent policy → decoupled PPO objective proposed in [1]

[1] Hilton, Jacob, Karl Cobbe, and John Schulman. "Batch size-invariance for policy optimization." *Advances in Neural Information Processing Systems* 35 (2022): 17086-17098.

Two challenges to make it asynchronous

Data Staleness:

The policy that generated the rollout might be too old for the current training policy (off-policy issue)

Solution: don’t start new rollout task until the current tasks+predefined staleness tolerance have been consumed(used in training)

Inconsistent policy:

Policies used for a single rollout may contains multiple different policies

Solution: decoupled PPO objective