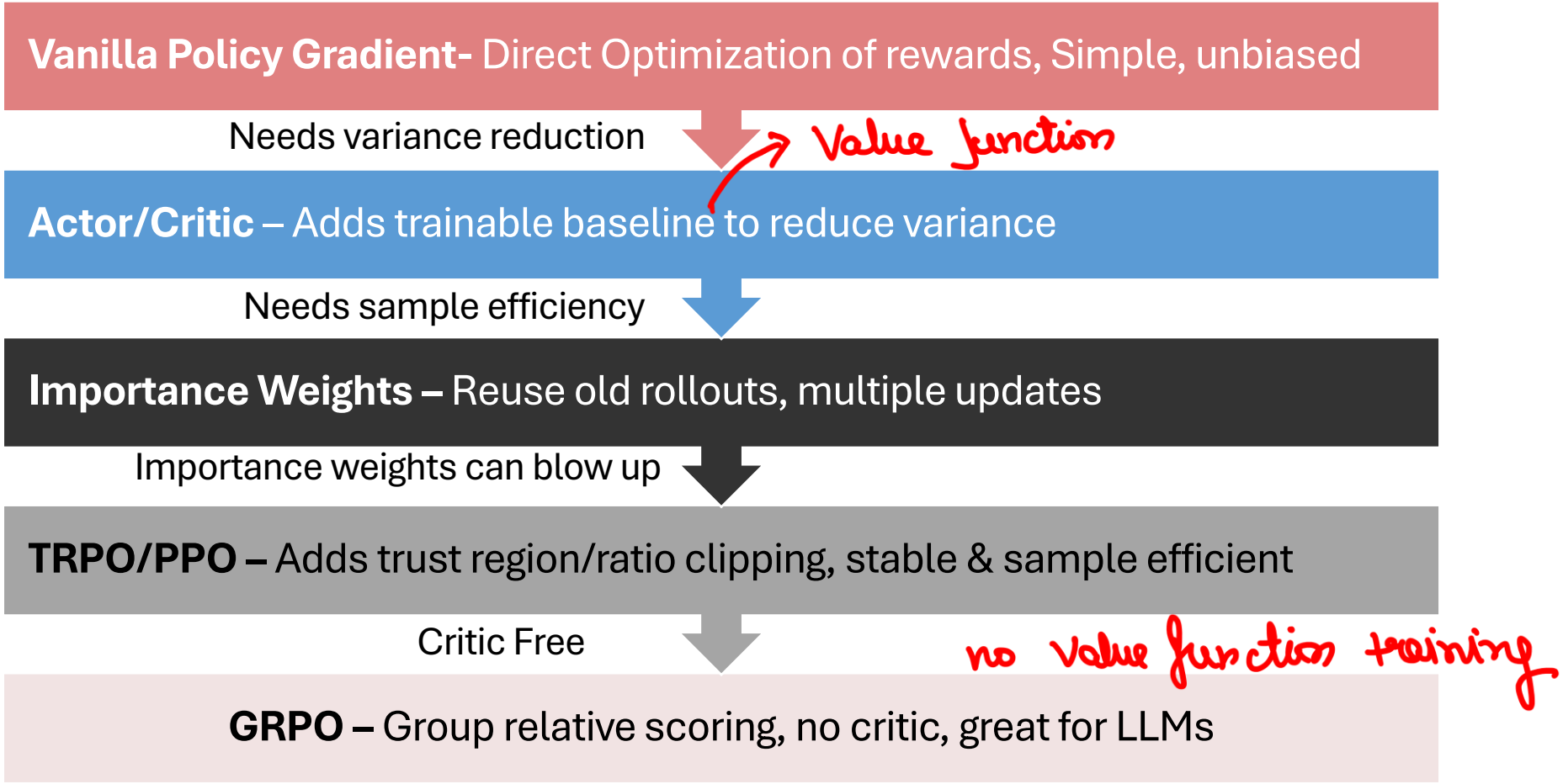
Alignment of Language Models – Reward Maximization

Advances in Large Language Models

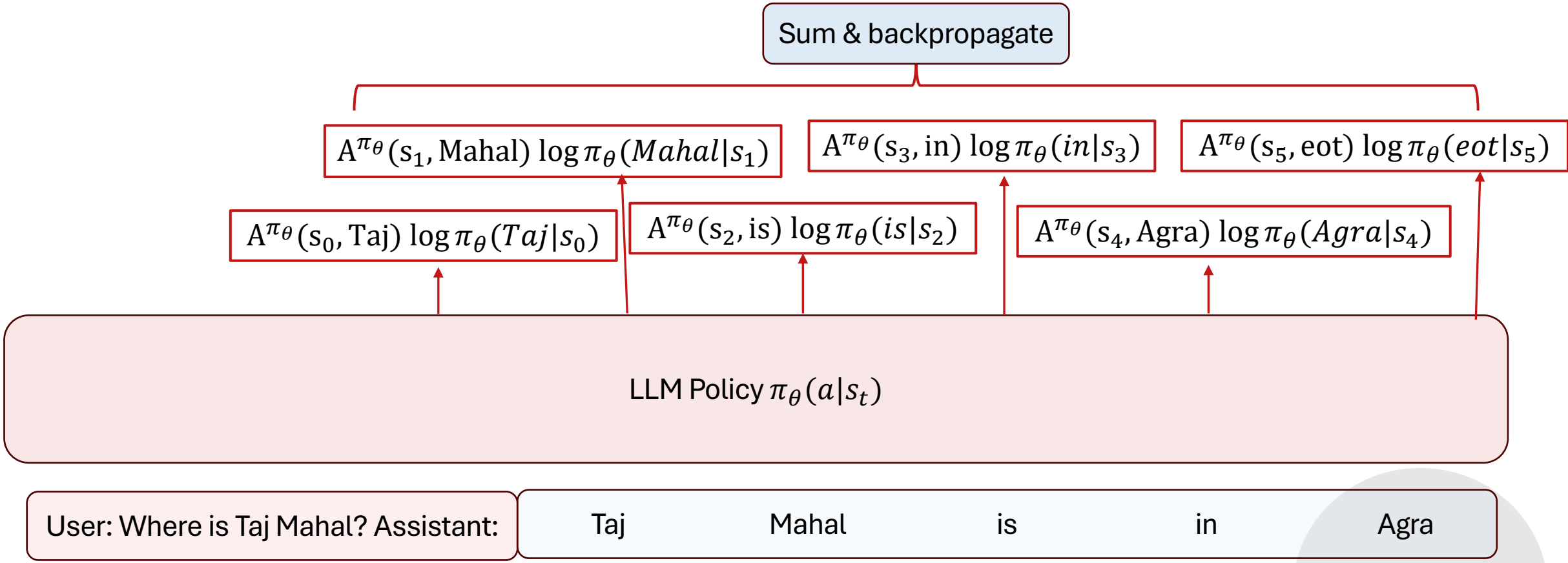
ELL8299 · AIL861

Gaurav Pandey
Senior Research Scientist, IBM Research

Journey from Vanilla PG to GRPO

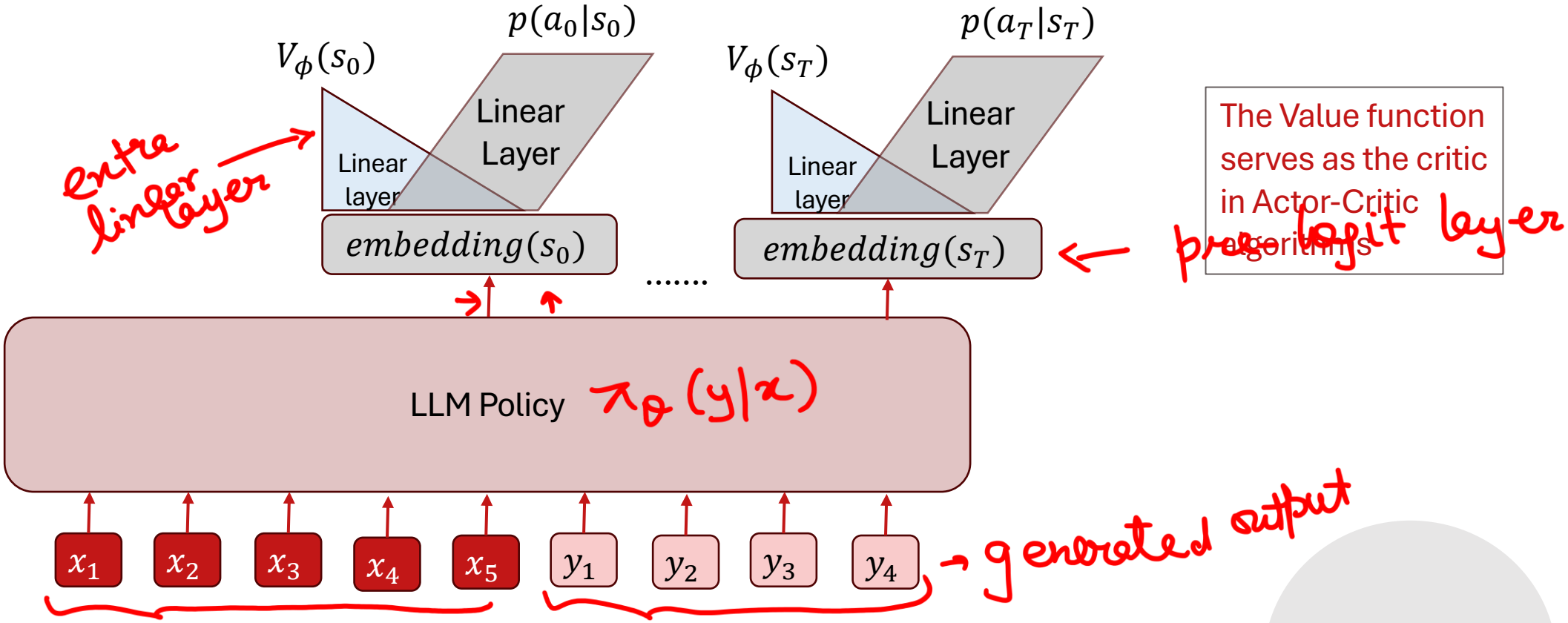


REINFORCE with advantage functions



Doesn't matter what gets generated in the future. The "reward" at token "Taj" is fixed.

Estimating the value function



$$Loss = E\left[\left(V_\phi(s_t) - R_t\right)^2\right] \text{ where } \mathrm{R_t} = r(s_t, a_t) + \gamma\, r(s_{t+1}, a_{t+1}) + \cdots + \gamma^{T-t}\, r(s_T, a_T)$$

Estimating the advantage function - I

- Difference between Q-function and Value function

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

→ ≈ LLM V_ϕ

Full Monte-carlo estimate

- Let $y = (a_0, \ldots, a_T) \sim \pi_\theta(y|x)$

$$\hat{Q}^{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \ldots + \gamma^{T-t} r(s_T, a_T)$$

($\gamma = 1$, Q = sum of future rewards)

$$\hat{V}^{\pi_\theta}(s_t) = V_\phi(s_t)$$

- This estimate is unbiased but very noisy.

Estimating the advantage function - II

- Difference between Q-function and Value function

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - V^{\pi_\theta}(s_t)$$

The Temporal Difference estimate

- Let $y = (a_0, \ldots, a_T) \sim \pi_\theta(y|x)$

$(s_t, a_t) \rightarrow$ new state $= s_{t+1}$

$$\hat{Q}^{\pi_\theta}(s_t, a_t) = r(s_t, a_t) + \gamma V_\phi(s_{t+1})$$ → long-term

$$\hat{V}^{\pi_\theta}(s_t) = V_\phi(s_t)$$ ✓

$s_t = \hat{Q} - \hat{V}$

- This estimate is biased but has low variance.

Multi-step Tradeoff

Let

$$\delta_t = r(s_t, a_t) + \gamma V_\phi(s_{t+1}) - V_\phi(s_t)$$

- 1-step advantage

$$\hat{A}^{(1)}(s_t, a_t) = \delta_t$$ ✓ lowest variance

- 2-step advantage

$$\hat{A}^{(2)}(s_t, a_t) = \delta_t + \gamma \delta_{t+1}$$

- 3-step advantage

$$\hat{A}^{(3)}(s_t, a_t) = \delta_t + \gamma \delta_{t+1} + \gamma^2 \delta_{t+2}$$

- Full Monte Carlo

$$\hat{A}^{(T-t)}(s_t, a_t) = \delta_t + \gamma \delta_{t+1} + \ldots + \gamma^{T-t} \delta_T$$

✓ lowest bias

Generalized Advantage Estimate (GAE)

- Exponential weighted average of the k-step advantages

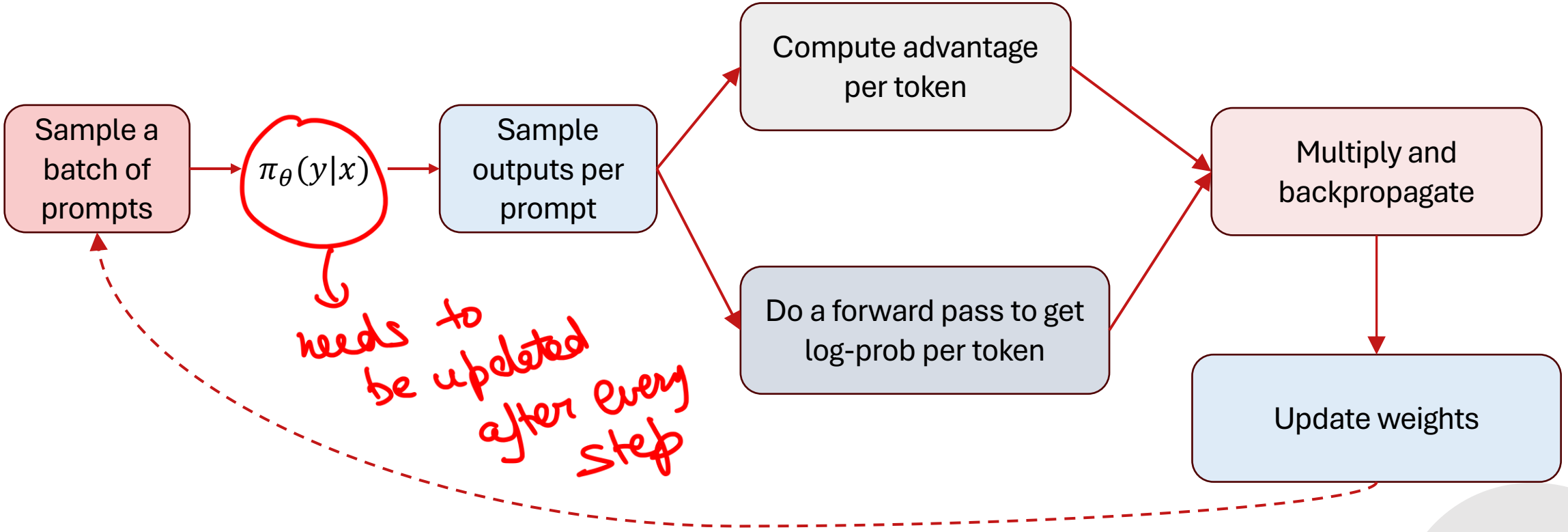
$$\hat{A}^{GAE}(s_t, a_t)$$

$$= \frac{\left(\hat{A}^{(1)}(s_t, a_t) + \lambda\, \hat{A}^{(2)}(s_t, a_t) + \lambda^2\, \hat{A}^{(3)}(s_t, a_t) + \cdots + \lambda^{T-t-1}\, \hat{A}^{(T-t)}(s_t, a_t)\right)}{1 + \lambda + \ \ldots + \lambda^{T-t-1}}$$

$$= \delta_t + (\lambda\gamma)\delta_{t+1} + \ \ldots + (\lambda\gamma)^{T-t}\, \delta_T$$

- Typically, $\gamma = 1\ and\ \lambda = 0.95$

The story so far



The samples are used once and never again.
Highly wasteful!!

The problem of data efficiency

- Every time we update the policy
  - The distribution π_θ changes
  - The samples per prompt are outdated.
  - The samples need to be thrown away – they are not on-policy anymore.
  - Highly wasteful – Sampling is expensive.
  - Communicating the updated weights to the sampler is expensive.
- Can we squeeze more juice out of the samples before throwing them away ?

Reusing old samples

- Let $y = (a_0, \ldots, a_T)$ be a sample from $\pi_\theta(y|x)$
- The gradient of the training objective is

$$\sum_{t=0}^{T} \hat{A}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- What if $y = (a_0, \ldots, a_T)$ was a sample from a previous policy $\pi_{\theta_{old}}(y|x)$?

Introducing importance weights

- Suppose, you wish to compute $E_{x\sim p}[f(x)]$
- But you can only sample from q

$$E_{x\sim p} f(x) = \sum_{x\in X} p(x) f(x) \times \frac{q(x)}{q(x)}$$

$$= \sum_{x\in X} q(x) \left(\frac{p(x)}{q(x)}\right) f(x)$$

$$= E_{x\sim q}\left[\frac{p(x)}{q(x)}\right] f(x)$$

Importance Weights

Gradient estimation with old samples

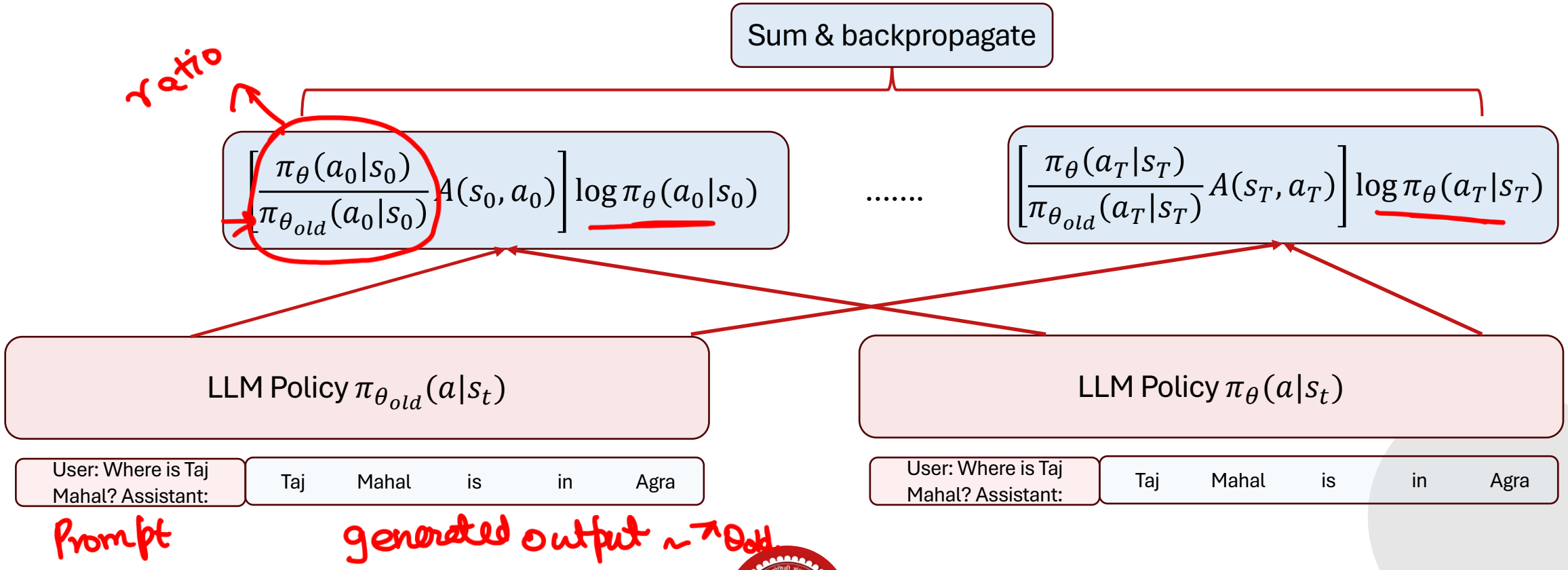
- Let $y = (a_0, \ldots, a_T)$ be a sample from $\pi_{\theta_{old}}(y|x)$
- The gradient of the training objective is

$$\sum_{t=0}^{T} \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \right) \hat{A}(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t)$$

- $r_t(\theta) = \left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \right)$ are the per-step importance weights

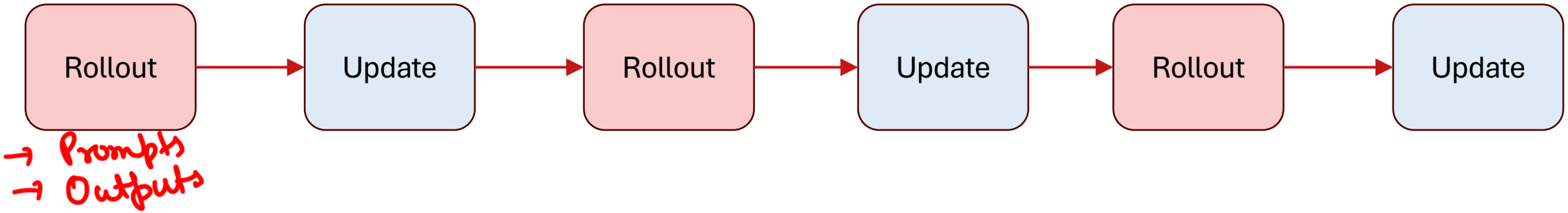
REINFORCE estimate with importance weights

The term in the square brackets is kept constant during backpropagation. In Pytorch, this means using .detach() function



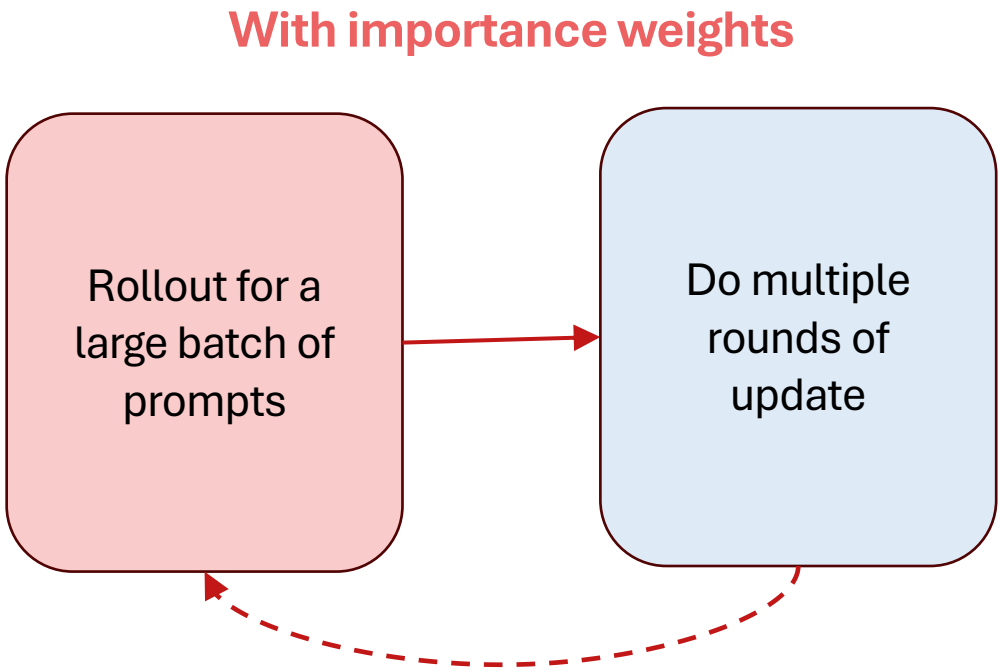
Why importance weights are useful?

Without importance weights



- Rollout – Sampling from the latest policy (using vllm or sglang)
- Update – Update the weights based on the rollout samples
- Very expensive.
- Unstable training dynamics

Why importance weights are useful?



- Rollout – Sampling from an old policy (vllm or sglang)
- Update – Update the weights based on the rollout samples

How to use this large rollout?

- How does this dataset look like?
- For each prompt x in batch

$$(x, y_1), \ldots, (x, y_m)$$

- If the rollout batch is too large
  - Expensive gradient update
  - Overfitting

- **Question**: How to train on large rollout batches?

Mini-batches

How to use this large rollout?

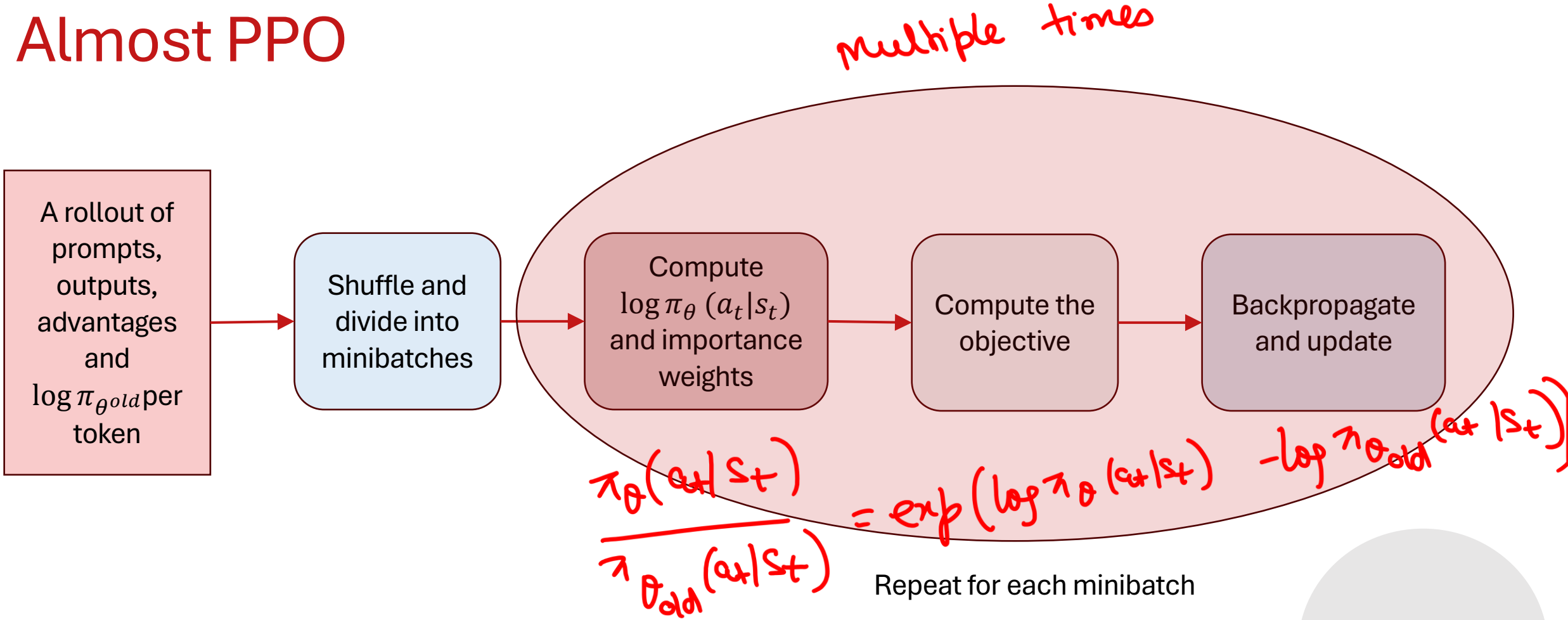
- How does this dataset look like?
- For each prompt x in batch

$$(x, y_1), \ldots, (x, y_m)$$

- For each (x, y_i)
  - Compute advantage per token
  - Compute $\log \pi$ per token

$\log \pi_{\theta_{old}}(a_t|s_t)$

Almost PPO

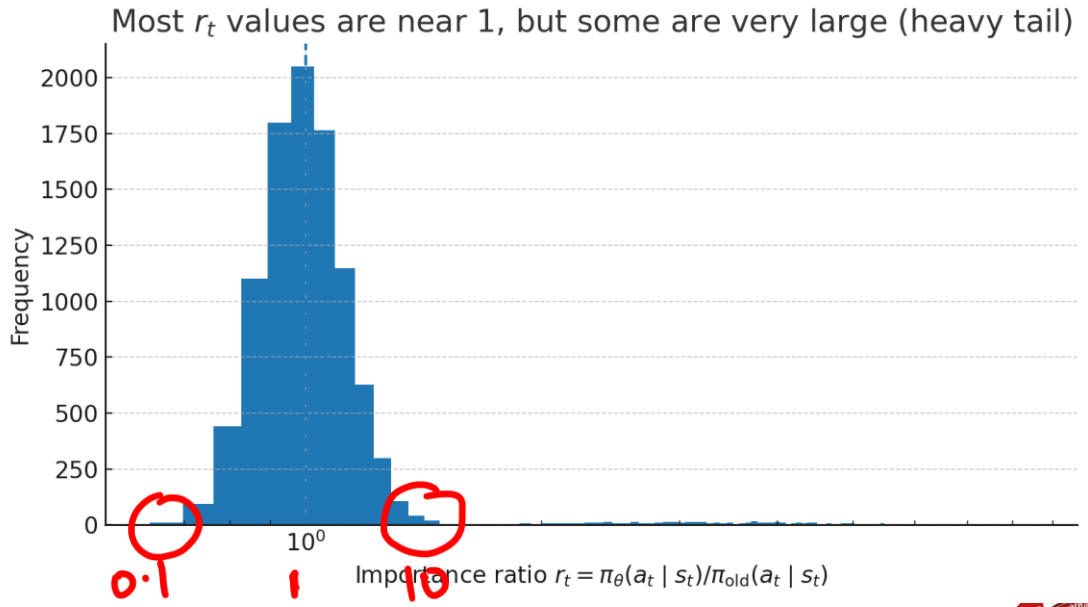


Do multiple epochs of training on this rollout.

The problem with importance weights

$$r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

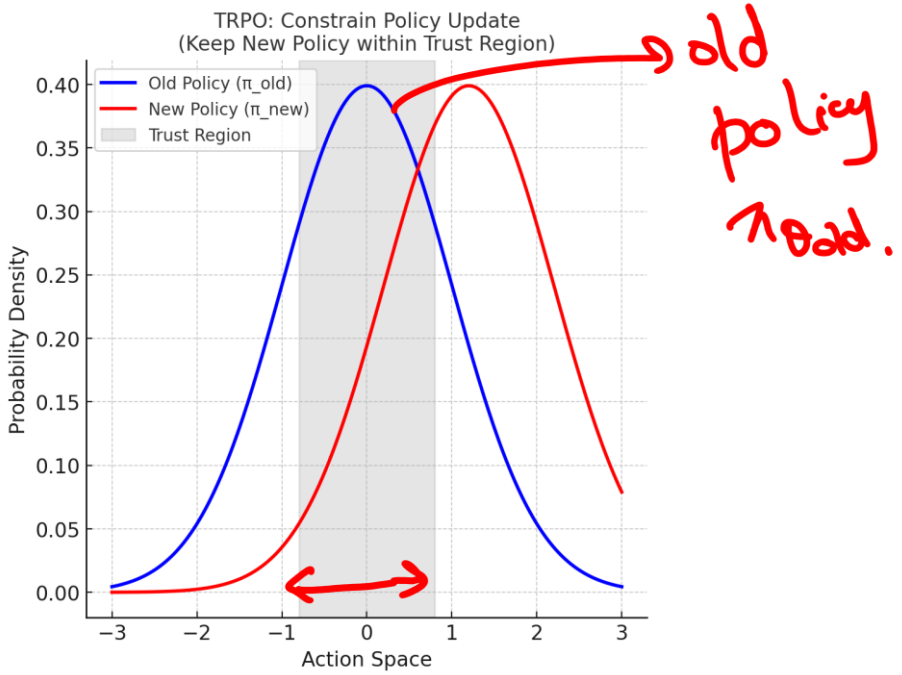
- If π_θ drifts far, ratios explode



Leads to high variance and destructive updates

Trust Region: Keep policies close

- Idea: Constrain update so that $KL(\pi_\theta || \pi_{\theta_{old}}) < \delta$



- Pros: Stable update
- Cons: Computationally expensive (requires second-order updates)
- This is called Trust Region Policy Optimization

PPO = TRPO made simple

- Original objective: Importance-weighted advantage

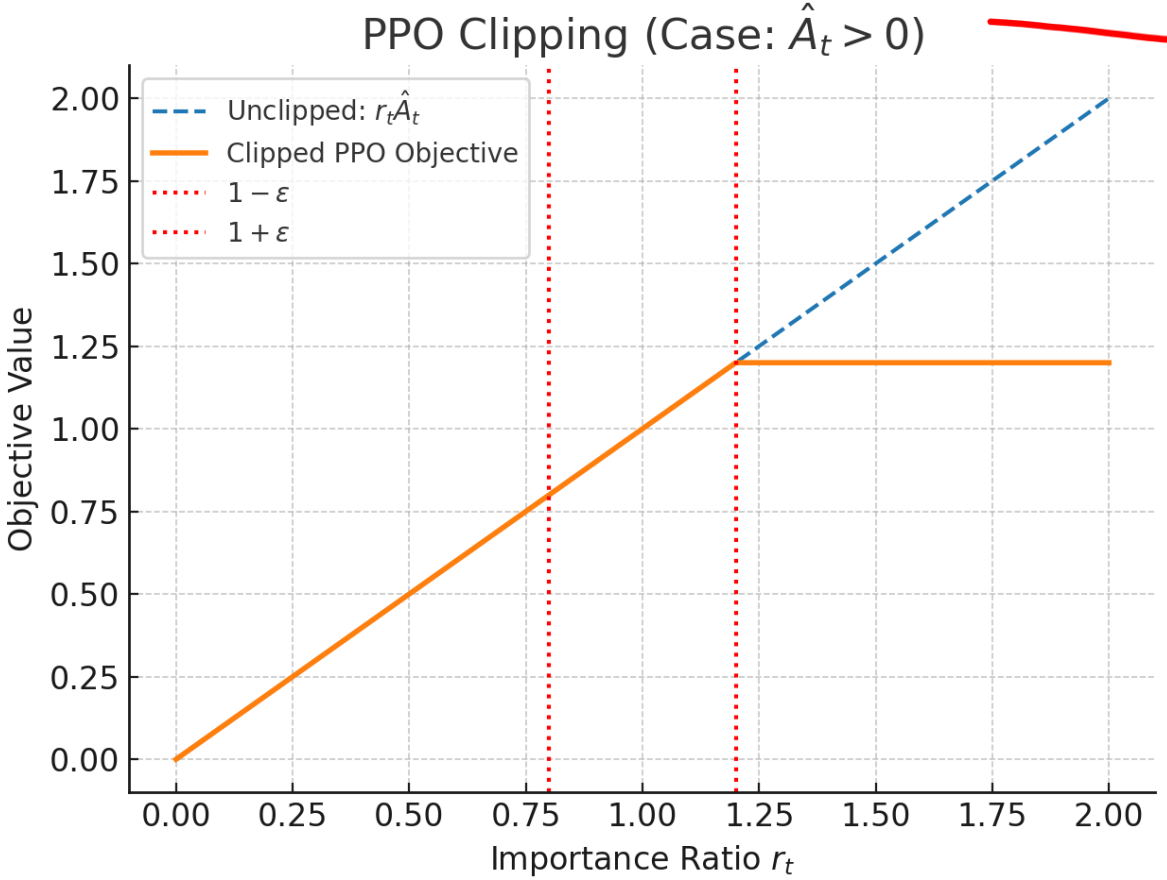
$$\hat{A}_t = \hat{A}(s_t, a_t)$$

$$L(\theta) = E\left[r_t(\theta)\hat{A}_t\right]$$

- PPO objective

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

Why Clipping Works (Case 1: Positive Advantage)

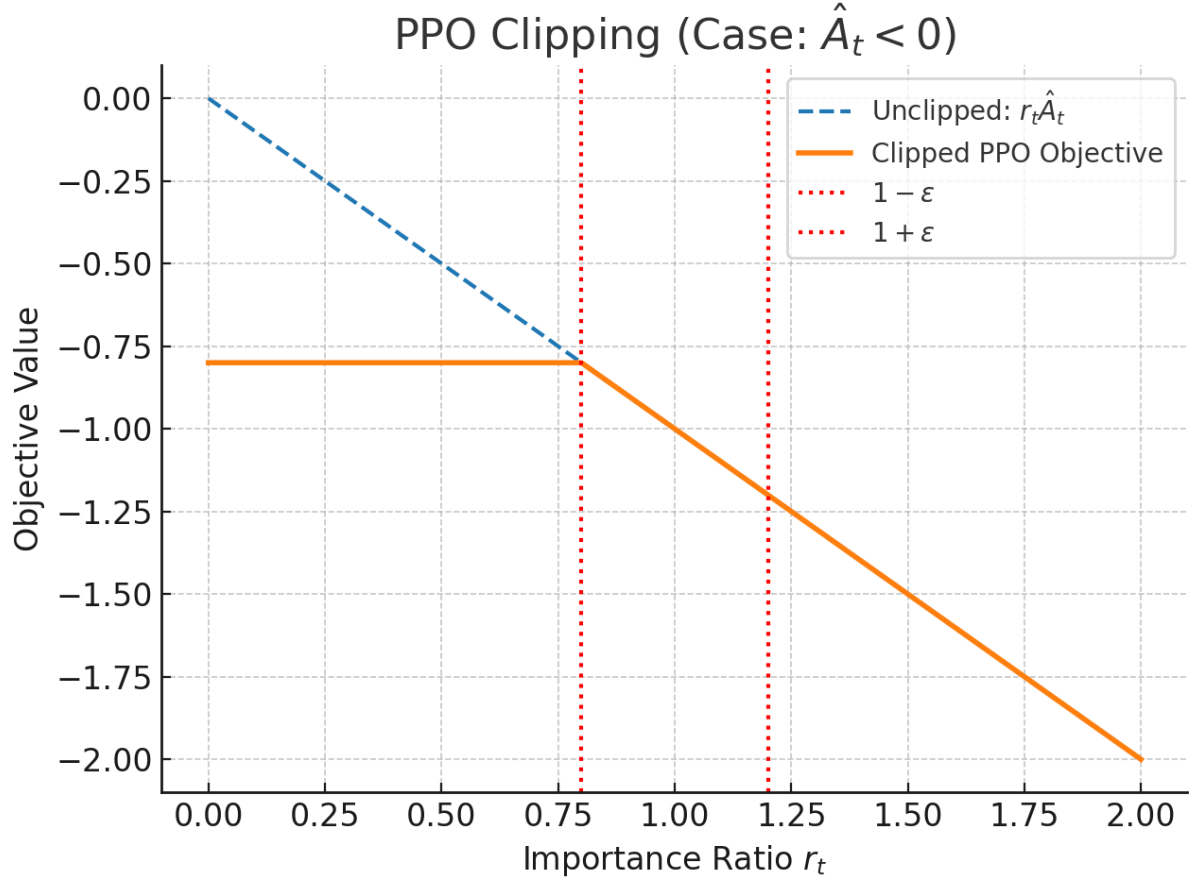


→ +ve sample

→ Increase prob of sample

→ But not beyond a limit.

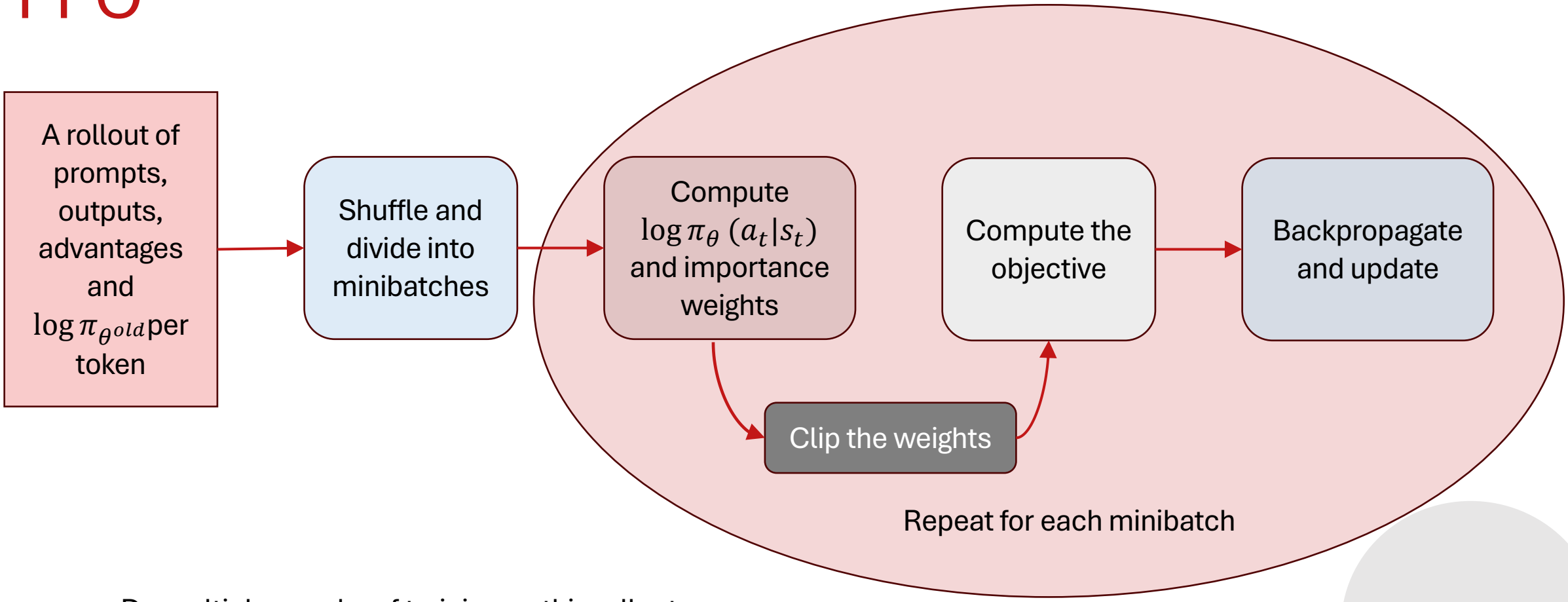
Why Clipping Works (Case 2: Negative Advantage)



Intuition behind clipping

- $r_t(\theta)$ = How far you can push an action's probability?
- Clipping prevents overshoot in either direction
- Keeps updates conservative
- Still allow multiple updates per rollout
- Sample efficiency + stability

PPO



Do multiple epochs of training on this rollout.

From Policy Gradient - PPO

- Vanilla Policy Gradient – Unstable, high variance
- Actor-Critic – Variance Reduction with Q/Value/Advantage estimation
- Importance Weights – Reuse rollouts, multiple epochs
- Problem: Ratios blow up
- Solution: PPO clipping (simple, stable, efficient)

Why look beyond PPO?

- Problems with PPO:
  - Critic/Value function – extra network to train
  - Value estimation bias/instability can hurt performance
- Desirable:
  - Simpler training objective (no critic)
  - Use relative feedback within a batch (instead of absolute values)

Core idea of GRPO

- Instead of learning $V_\phi(s)$, GRPO compares outputs within the same group of rollouts for a prompt.
- For each prompt x, sample K completions $\{y_1, \dots, y_K\}$
- Compute rewards $r(x, y_i)$
- Normalize reward within the group to get Advantages

all outputs are for the same input

$$A(y_i) = \frac{r(x, y_i) - mean\big(r(x, y_1), \dots, r(x, y_K)\big)}{stddev\big(r(x, y_1), \dots, r(x, y_K)\big) + \epsilon}$$

GRPO objective - PPO Without a Critic

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

Not a function of time

- No value function needed
- Advantage is relative
  - Outputs better than average are encouraged.
  - Outputs worse than average are discouraged.

GRPO – when and when not to use?

When to use:

- Cheap to generate K trajectories/outputs per prompt.
- Absolute reward is unreliable.
- The trajectories are short and stochasticity is low.
- **Eg: LLMs for response generation**

When not to use:

- Rollouts are slow.
- Trajectories are long – Need to do credit assignment
- **Eg: Games & robotics**

Journey from Vanilla PG to GRPO

Vanilla Policy Gradient- Direct Optimization of rewards, Simple, unbiased

Needs variance reduction

Actor/Critic – Adds trainable baseline to reduce variance

Needs sample efficiency

Importance Weights – Reuse old rollouts, multiple updates

Importance weights can blow up

TRPO/PPO – Adds trust region/ratio clipping, stable & sample efficient

Critic Free

GRPO – Group relative scoring, no critic, great for LLMs

Resources

Trust Region Policy Optimization – Schulman et al., 2015

High-Dimensional Continuous Control Using Generalized Advantage Estimation: Schulman et al., 2015

Proximal Policy Optimization Algorithms - Schulman et al., 2017

OpenAI Spinning Up (tutorials): Clear PPO notes + annotated PyTorch code.

GRPO (why + where it came from): Introduced in DeepSeekMath paper, 2024

Hugging Face TRL PPO Trainer guides & details