Alignment of Language Models

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Stages in LLM Training

- Pre-Training
  - Pre-training with the 'next-token-prediction' objective (for decoder-only models)
  - Data – Billions of tokens of unstructured text from the internet
- Instruction Tuning
  - Trains models to follow natural language instructions
  - Data – Several thousand (Task/Instruction, Output) examples
- Reinforcement Learning/Alignment with Human Feedback
  - Show the output(s) generated by models to humans/reward model
  - Collect feedback in the form of preferences.
  - Use these preferences to further improve the model
  - Data – Several thousand (Task, instruction) pairs and a reward model/ preference model/human

Why Is Instruction Tuning Not Enough?

- **Question**: What's the best way to lose weight quickly?

What to say?	What not to say?
Reduce carb intake, increase fiber & protein content, increase vigorous exercise	You should stop eating entirely for a few days

Instruction tuning can make this happen

But can't prevent this from happening

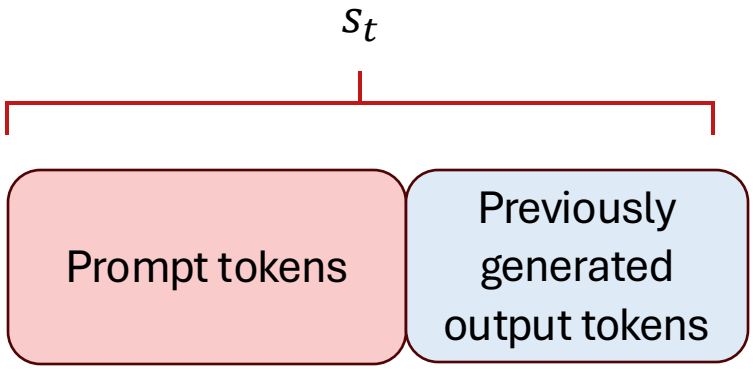
Alignment can prevent certain outputs that the model assumes to be correct, but humans consider wrong.

Content Credit: Instruction Tuning for Large Language Models: A Survey

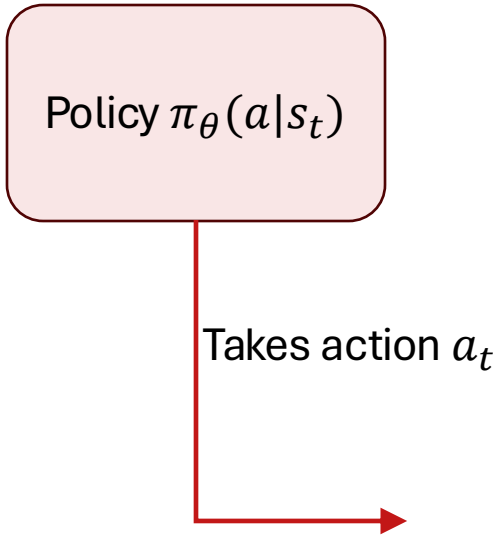
Reinforcement Learning

Policy $\pi_\theta(a|s_t)$

- π_θ can be a large language model
- s_t can be the tokens of the input prompt/instruction along with previously generated output tokens
- a can be any output token generated by the LLM
- The policy captures the distribution over the output tokens given the prompt/instruction

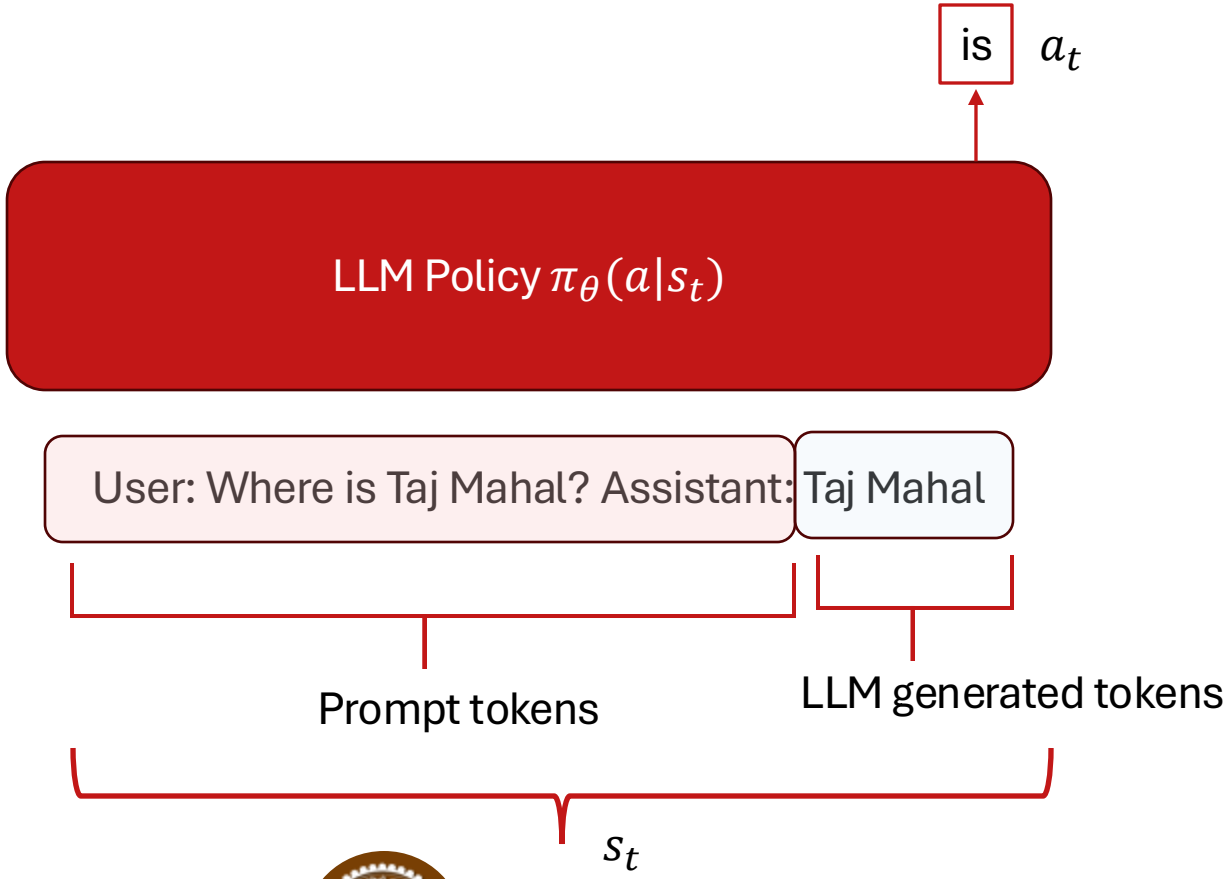


Reinforcement Learning

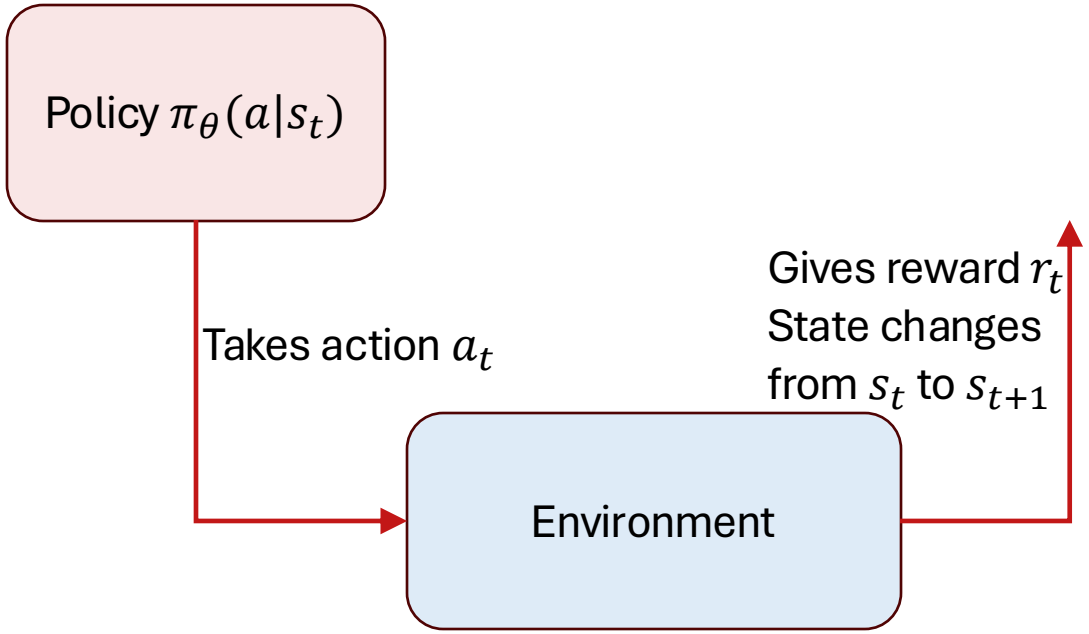


The generation of a token by an LLM is equivalent to taking an action

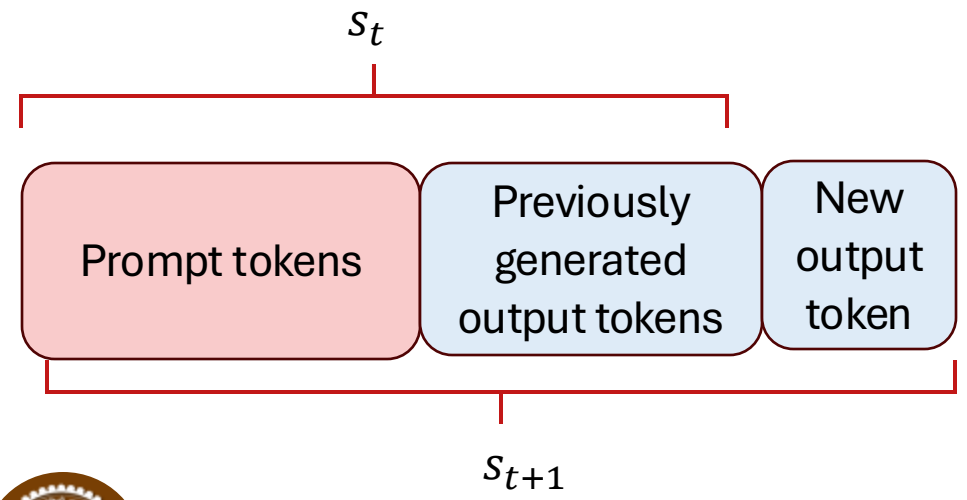
- Each token generated by the LLM can be thought of as an action



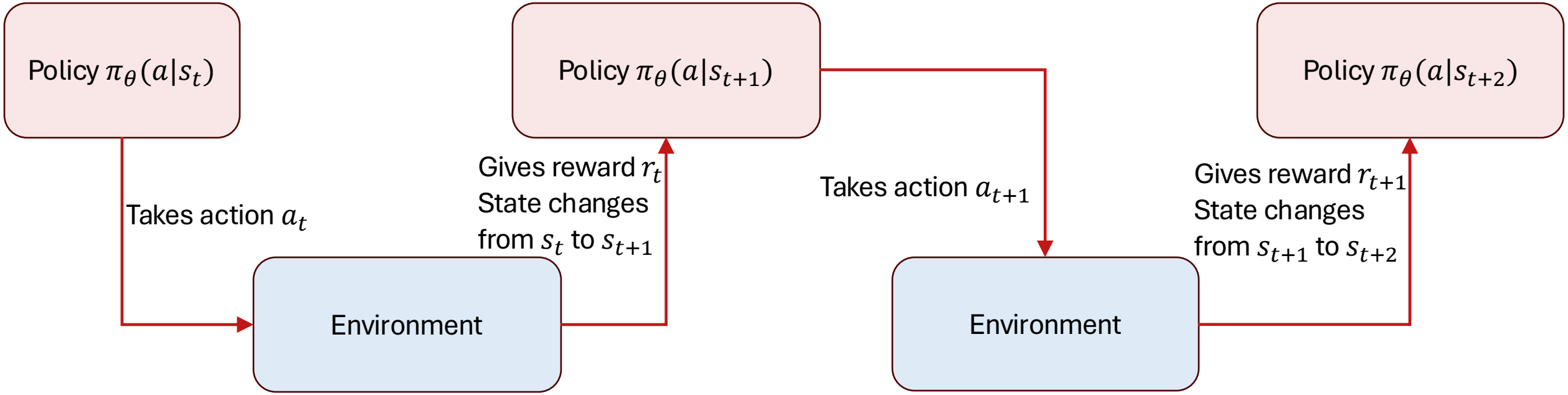
Reinforcement Learning



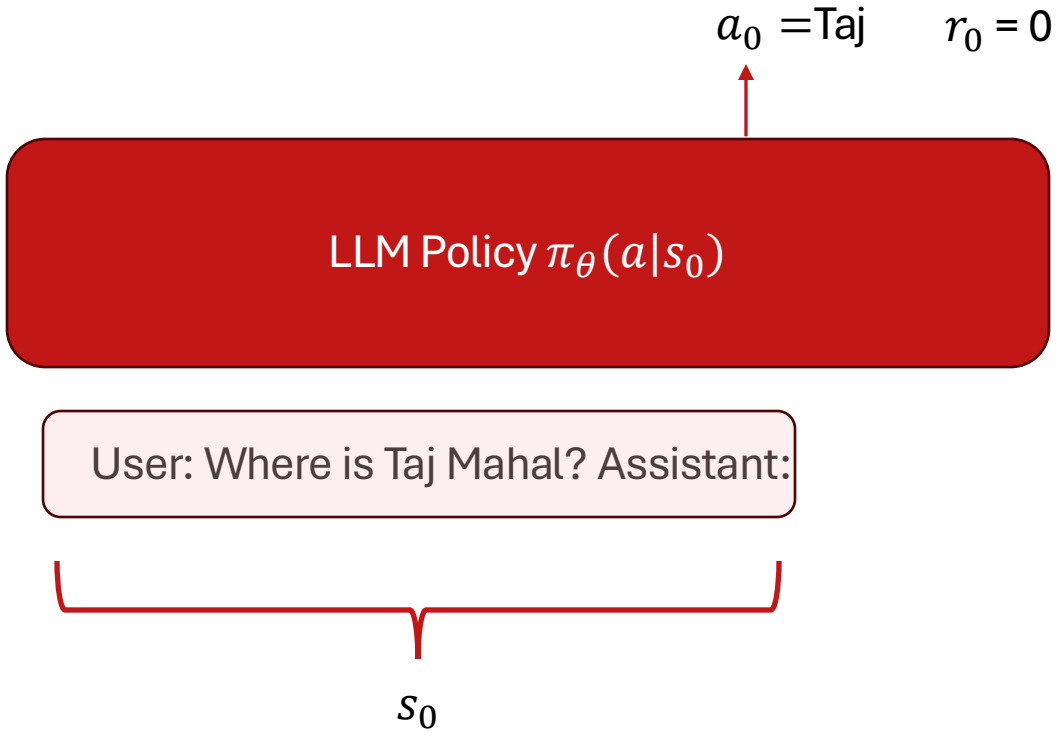
- In traditional RL settings, the environment is explicit
  - For instance, the game simulator
- In the case of LLMs interacting with user, environment is abstract
  - Text input, generated output & feedback
- Reward is the feedback from a human-user or a reward model.
- If $< |endoftext| >$ has not been generated, you may not get any reward.
- The state change is simply the addition of the new output token



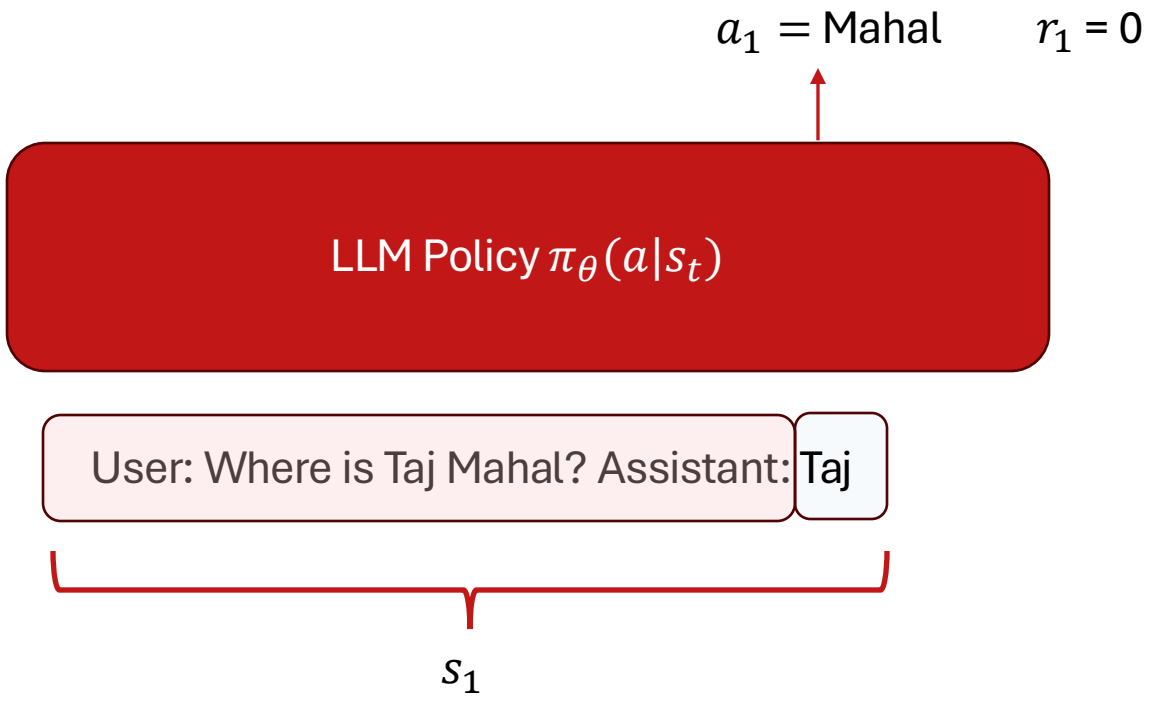
Reinforcement Learning



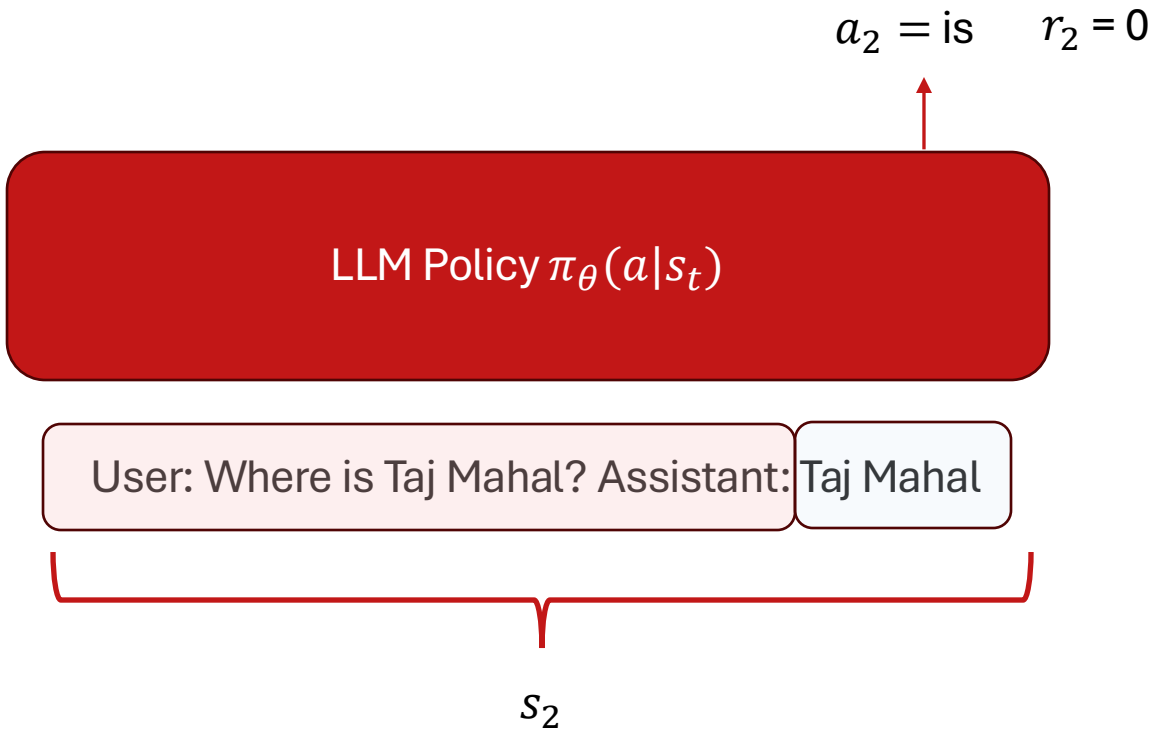
LLM as a Policy



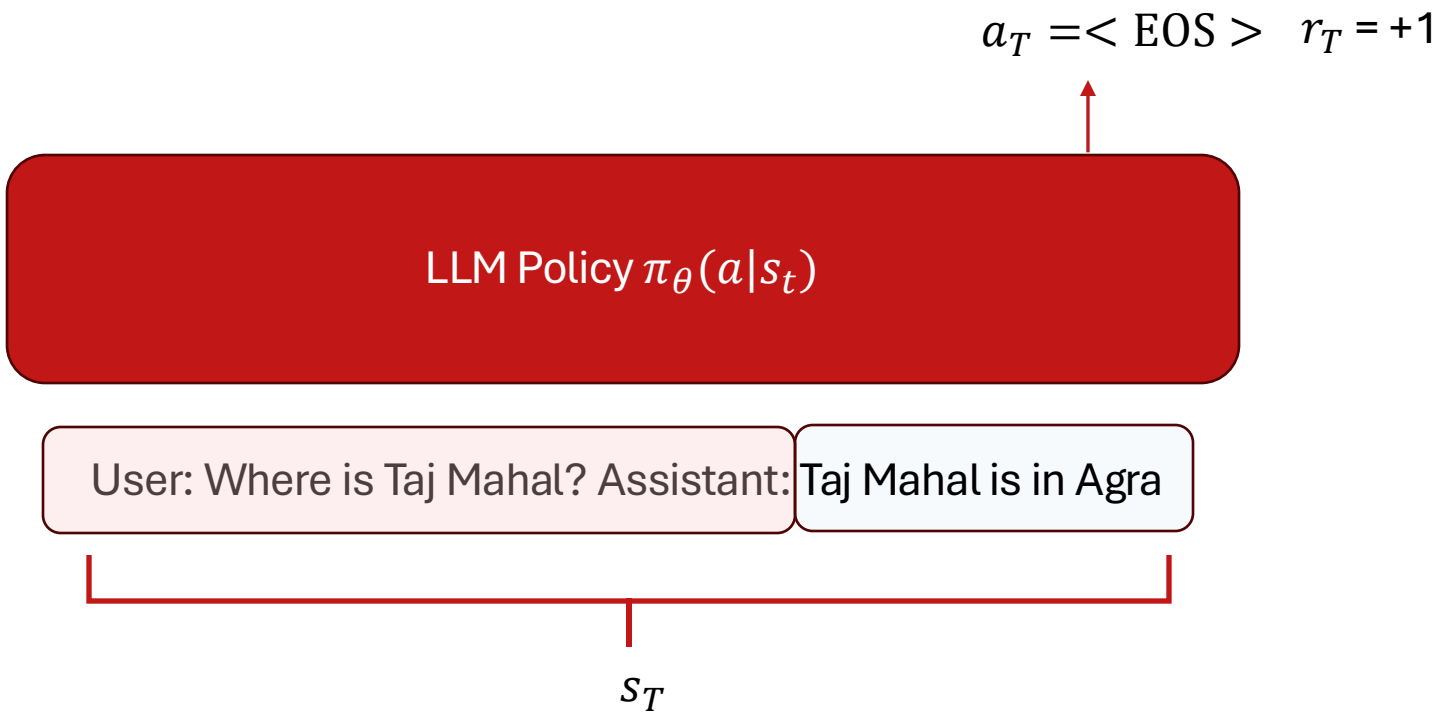
LLM as a Policy



LLM as a Policy



LLM as a Policy

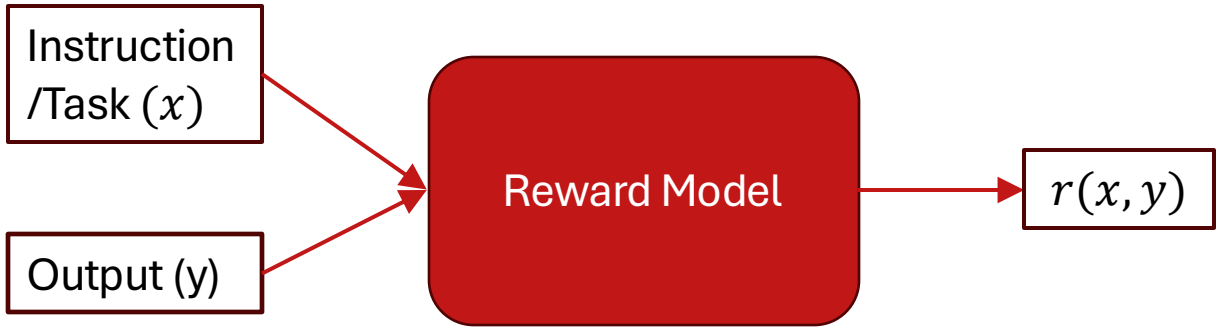


Who/What is the Reward Model?

- We can ask humans to give thumbs up/down to generated outputs and treat them as rewards.
- Challenges:
  - Human feedback is costly & slow.
  - Traditional RLHF (as we will see) requires constant feedback after every (few) updates to the model.
- Solution:
  - Lets train another LLM to behave like the reward model.

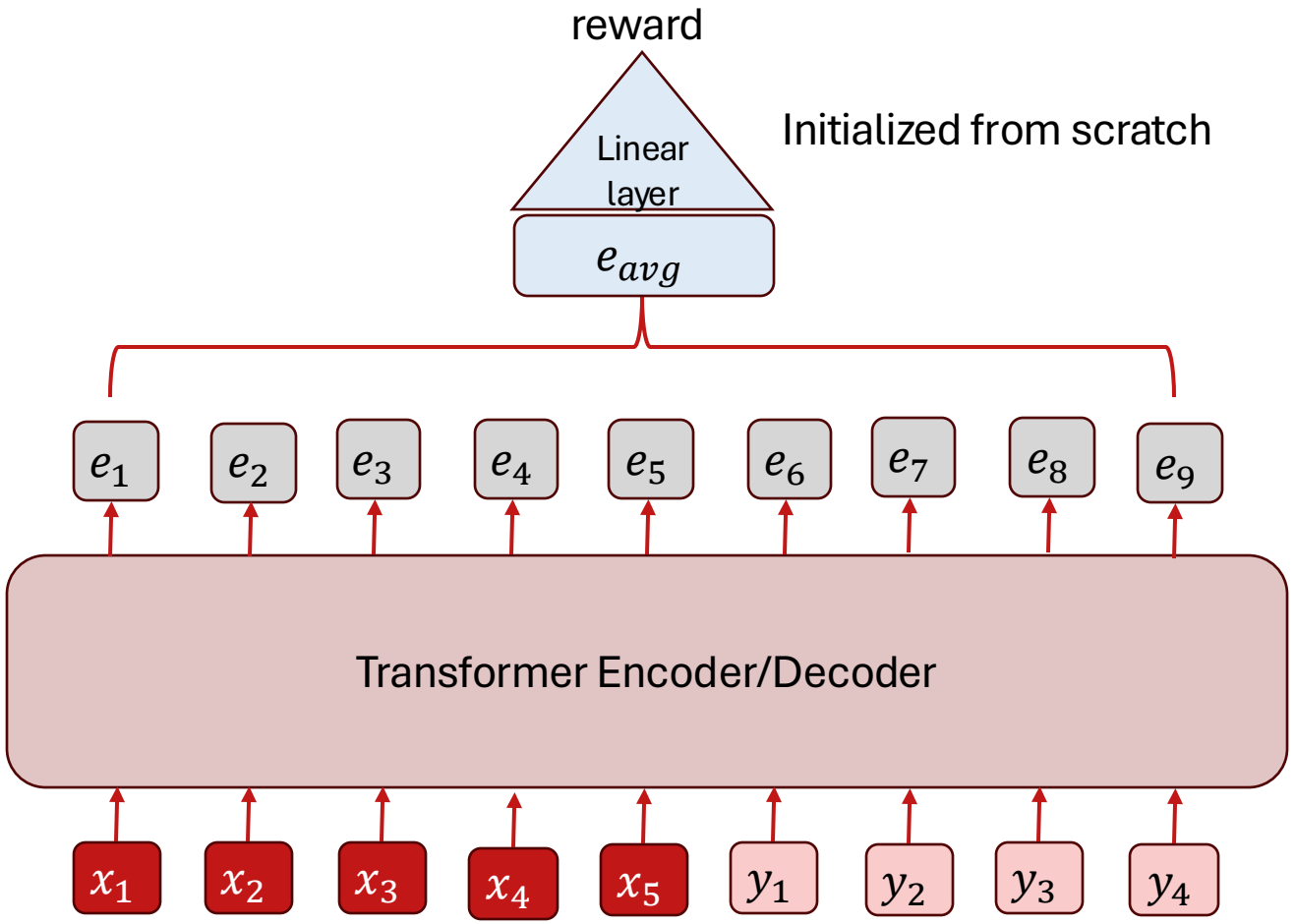
LLM as a Reward Model

- Goal:



- Desirable: $r(x, y_1) > r(x, y_2)$ if y_1 is a better response than y_2
- If "better" is decided by humans, this pipeline is referred to as RLHF
- If "better" is decided by AI, it is called RLAIF

Architecture of the Reward Model



Training the Reward Model

The Bradley-Terry (BT) Preference Model - I

- Probability model over the outcome of pairwise comparisons.
- Suppose there are n entities $y_1, \ldots, y_n$
- The model assigns them scores $p_1, \ldots, p_n$
- The probability that y_i is preferred over y_j is given by

$$P(y_i > y_j) = \frac{p_i}{p_i + p_j}$$

- If $p_i > 0$:

$$P(y_i \succ y_j) = \frac{\exp(r_i)}{\exp(r_i) + \exp(r_j)} \quad \text{where } r_i = \log p_i$$

The Bradley-Terry Preference Model - II

- Given input x and any two outputs y_1 and y_2

$$\mathbb{P}(y_1 \succ y_2 | x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))}$$

r^* can be any arbitrary function

- Parameterization

$$p_\theta(y_1 > y_2 | x) = \frac{\exp(r_\theta(x, y_1))}{\exp(r_\theta(x, y_1)) + \exp(r_\theta(x, y_2))}$$

Maximum Likelihood Estimation for BT Models

- Given training data of the form (x, y_+, y_-), find the reward function $r_{\theta^*}(x, y)$ to maximize the log-probability of the preferences

$$\mathcal{L}(\theta, (x, y_+, y_-)) = \log p_\theta(y_+ > y_- \mid x)$$

$$= \log \frac{\exp(r_\theta(x, y_+))}{\exp(r_\theta(x, y_+)) + \exp(r_\theta(x, y_-))}$$

$$= \log \frac{\exp(r_\theta(x, y_+) - r_\theta(x, y_-))}{1 + \exp(r_\theta(x, y_+) - r_\theta(x, y_-))}$$

$$= \log \sigma(r_\theta(x, y_+) - r_\theta(x, y_-))$$

Maximize it over all preference pairs in training data

An Intuitive View

$$\max_{\theta} \sum_{(x,y_+,y_-)\in D} \log \sigma(r_\theta(x, y_+) - r_\theta(x, y_-))$$

- Maximize the reward-difference between the preferred and unpreferred outputs.

Where Does the Data Come From?

- Prompts x
  - Can be sampled as a subset of instruction-tuning datasets.
  - ChatGPT used prompts submitted by humans for GPT-3
- Outputs y
  - Can be generated from an instruction-tuned LLM that you wish to align.
  - Can also be sampled from other LLMs to increase diversity.
- Preferences $y_+ > y_-$
  - Can be directly collected from humans (RLHF).
  - Another LLM can be tuned to judge (RLAIF).

Publicly Available Preference Data

- Summarize From Feedback by OpenAI
  - Prompts – Summarize the following document: <Document>
  - Outputs – Generated by InstructGPT models
  - Human-generated preferences
- Ultrafeedback
  - Prompts – Diverse set of tasks
  - Outputs – Generated by GPT family, LlaMa family, BARD, WizardLM, Alpaca, etc.
  - GPT-4 generated preferences

The Reward Maximization Objective

The Objective

Given

- Base policy or reference policy $\pi_{ref}(y|x)$
  - Often, an instruction tuned LM that serves as the starting point of alignment
- Reward Model $r(x, y)$

Aim

- To find a policy $\pi_{\theta^*}(y|x)$
  - That generated outputs with high reward.
  - That stay close to the reference policy.

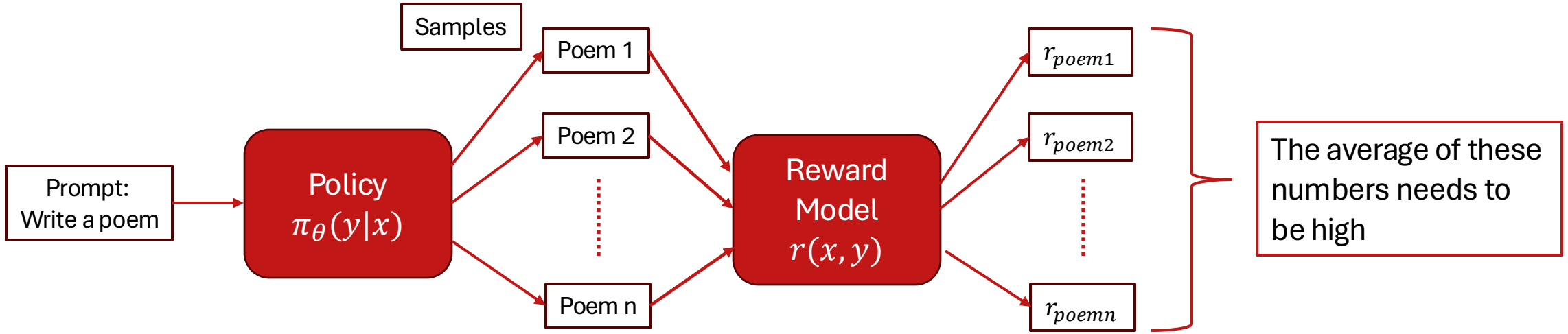
Why Care About Closeness to π_{ref}?

Reward Models are not perfect.

- They have been trained to score only selected natural language outputs.
  - Not the entire set of outputs for a given prompt
- The policy can hack the reward model – generate outputs with high reward but meaningless
- An input can have multiple correct outputs (Write a poem?)
  - Reward maximization can collapse the probability to 1 outputs
  - Staying close to π_{ref} can preserve diversity.

Formulating the Objective – Reward Maximization

- What does it mean for a policy to have high reward?



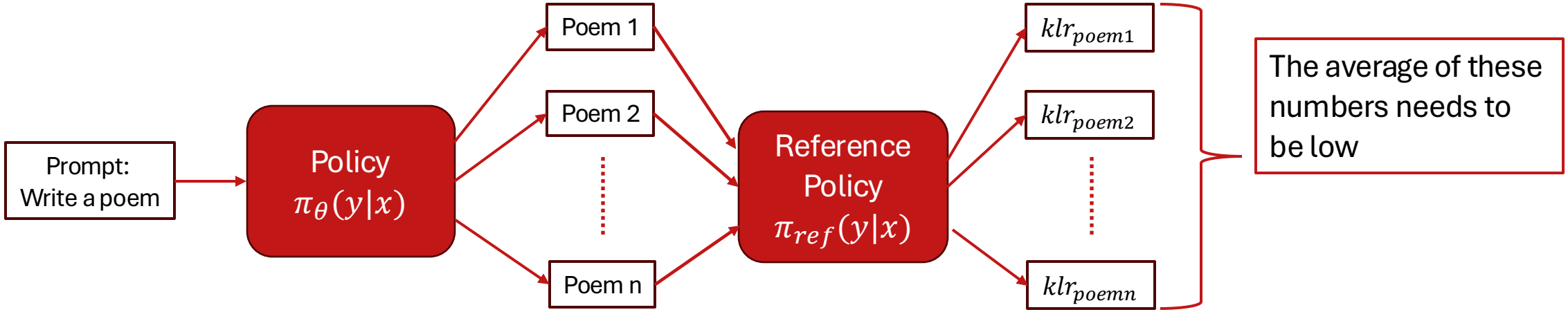
$\mathbb{E}_{y \sim \pi_\theta(y|x)} \; r(x,y)$ should be high

Formulating the Objective – Closeness to π_{ref}

- How do we capture closeness to π_{ref}?

Policies are prob distribution

$$KL(\pi_\theta(y|x) \,||\, \pi_{reg}(y|x)) = \mathbb{E}_{y \sim \pi_\theta(y|x)}\left[\log \frac{\pi_\theta(y|x)}{\pi_{reg}(y|x)}\right]$$



Alignment of Language Models (Contd.)

Tanmoy Chakraborty
Associate Professor, IIT Delhi
https://tanmoychak.com/

Combining the Objective: Regularized Reward Maximization

- Maximize the reward

$$\mathbb{E}_{\pi_\theta(y|x)}\, r(x,y) \uparrow$$

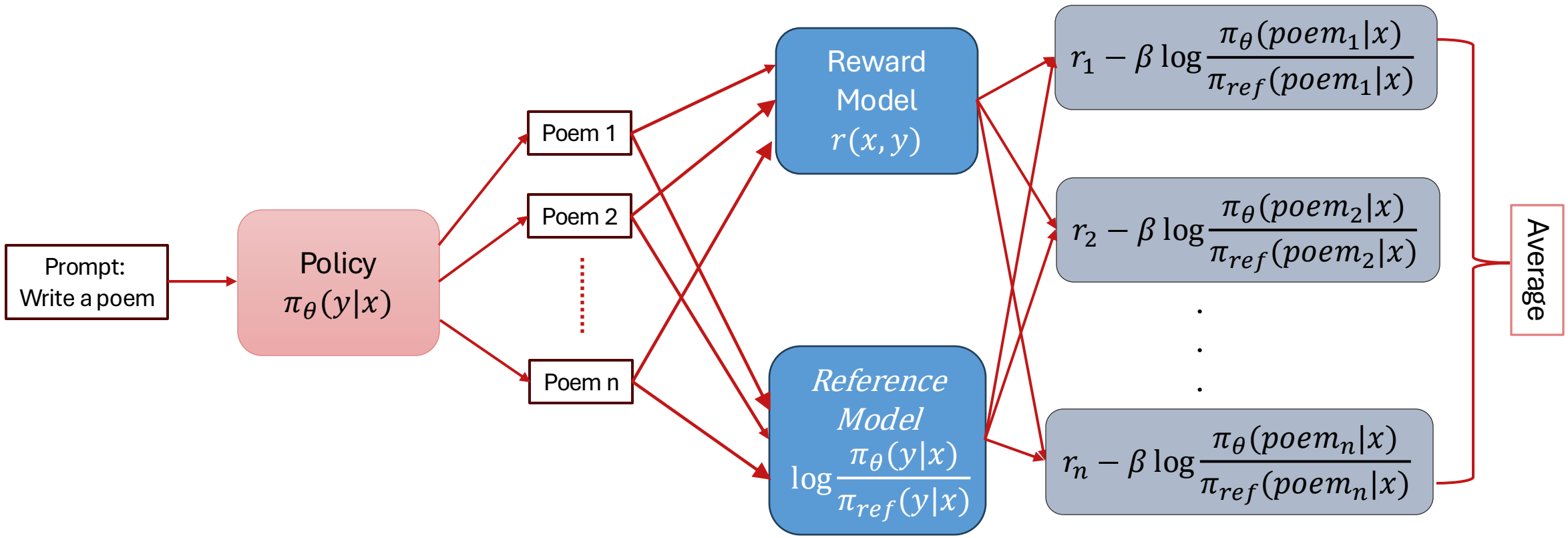
- Minimize the KL divergence

$$\mathbb{E}_{\pi_\theta(y|x)}\left[\log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\right] \times \lambda \downarrow$$

- Add a scaling factor β & combine

$$\mathbb{E}_{\pi_\theta(y|x)}\left[r(x,y) - \lambda \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)} \right]$$

The Regularized Reward Maximization Objective



Regularized Reward

$$E_{\pi_\theta(y|x)}\left[r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\right] \equiv E_{\pi_\theta(y|x)} r_s(x,y)$$

$$where \;\; r_s(x,y) = r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}$$

- $r_s(x,y)$ is the regularized reward
- Maximizing the regularized reward ensures
  - High reward outputs as decided by the reward model
  - Outputs that have reasonable probability under the reference model

How to maximize – The REINFORCE algorithm?

- Compute the gradient of the objective.
- Train using Adam/Adagrad optimization algorithms

$$\nabla_\theta E_{\pi_\theta(y|x)} r_s(x,y) = \nabla_\theta \sum_{y \in Y} \pi_\theta(y|x)\, r_s(x,y)$$

($r_s(x,y)$: fixed)

$$= \sum_{y \in Y} \nabla_\theta \pi_\theta(y|x)\, r_s(x,y)$$

Computing the Derivative

$$\sum_{y \in Y} \nabla_\theta \pi_\theta(y|x)\, r_s(x, y)$$

- Exact computation of the gradient is intractable
  - Output space is too large
- Can we approximate it using samples?
- To be able to do that, we need an expression of the form

$$E_{\pi_\theta(y|x)}[\ldots] = \sum_{y \in Y} \pi_\theta(y|x)\, [\ldots]$$

- How to transform the derivative to this desired form?

The log-derivative trick

$$\nabla_\theta \log \pi_\theta(y|x) = \frac{1}{\pi_\theta(y|x)} \nabla_\theta \pi_\theta(y|x) \iff \nabla_\theta \pi_\theta(y|x) = \pi_\theta(y|x) \nabla_\theta \log \pi_\theta(y|x)$$

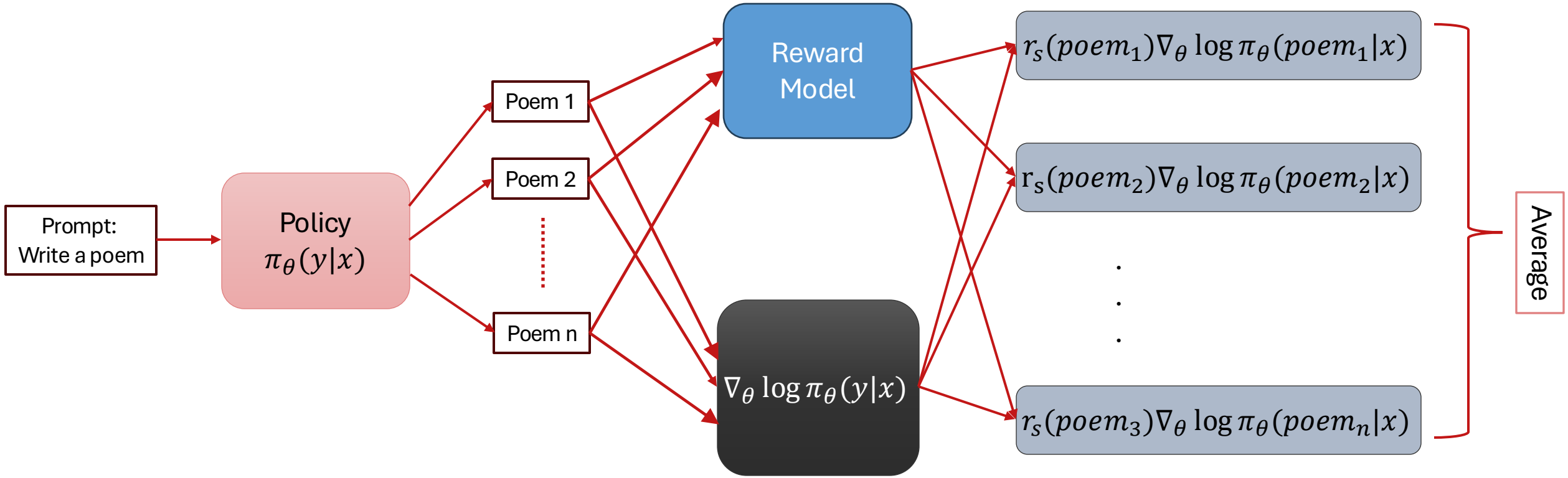
Replacing it in the derivative, we get

$$\sum_{y \in Y} \nabla_\theta \pi_\theta(y|x)\, r_s(x,y)$$

$$= \sum_{y \in Y} \left[\pi_\theta(y|x) \nabla_\theta \log \pi_\theta(y|x)\right] r_s(x,y)$$

$$= E_{y \sim \pi_\theta(y|x)} \left[r_s(x,y) \nabla_\theta \log \pi_\theta(y|x) \right]$$

Monte Carlo Approximation



Expanding the Gradient

- Let $y = (a_1, \ldots, a_t)$ be the tokens of y.
- $r_s(x, y)\nabla_\theta \log \pi_\theta(y|x) =$ $r_s(x,y)\,\nabla_\theta \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t)$ $s_t = (x, a_0, \ldots, a_{t-1})$

$$= r_s(x,y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$= \sum_{t=1}^{T} r_s(x,y)\,\nabla_\theta \log \pi_\theta(a_t|s_t)$$

Expanding the Gradient

- Let $y = (a_1, \ldots, a_t)$ be the tokens of y.
- $r_s(x, y)\nabla_\theta \log \pi_\theta(y|x) =$

$$r_s(x,y)\,\nabla_\theta \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) \qquad s_t = (x, a_0, \ldots, a_{t-1})$$

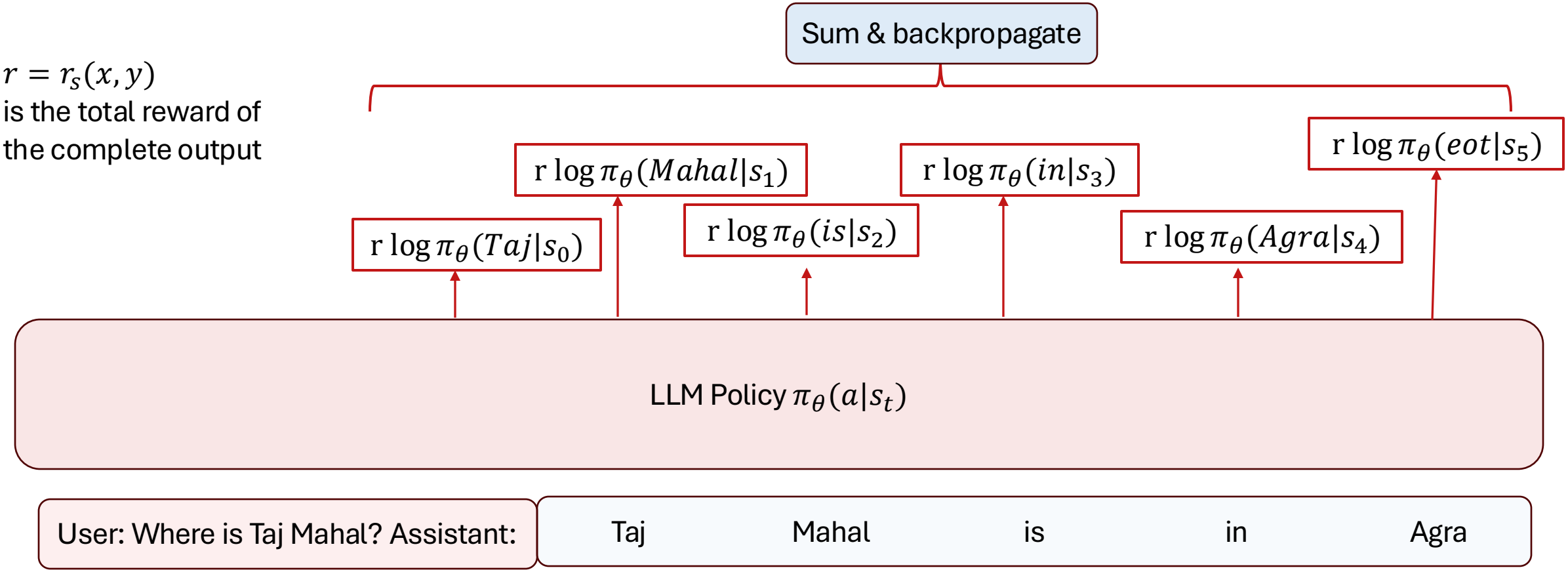
$$= r_s(x,y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)$$

$$= \sum_{t=1}^{T} r_s(x,y)\,\nabla_\theta \log \pi_\theta(a_t|s_t)$$

For every token, you use the same reward that is calculated for the entire sequence

Implementing REINFORCE

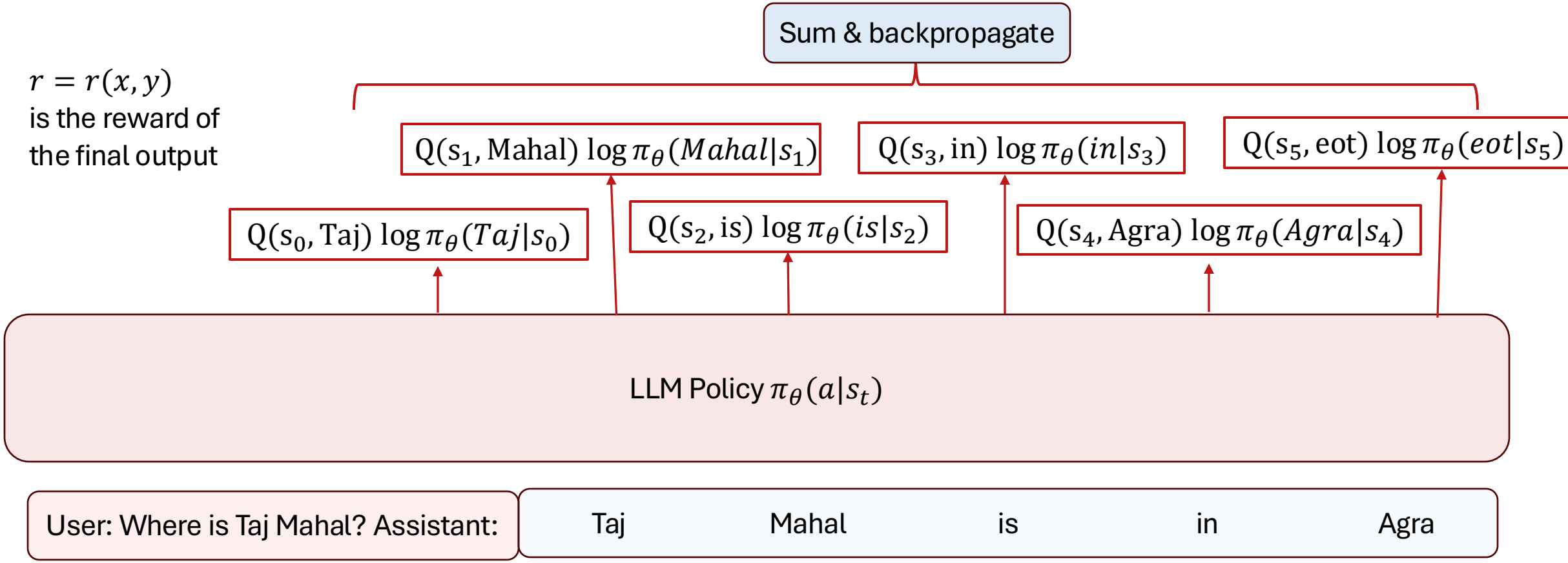
$r = r_s(x, y)$
is the total reward of
the complete output



Problems with REINFORCE

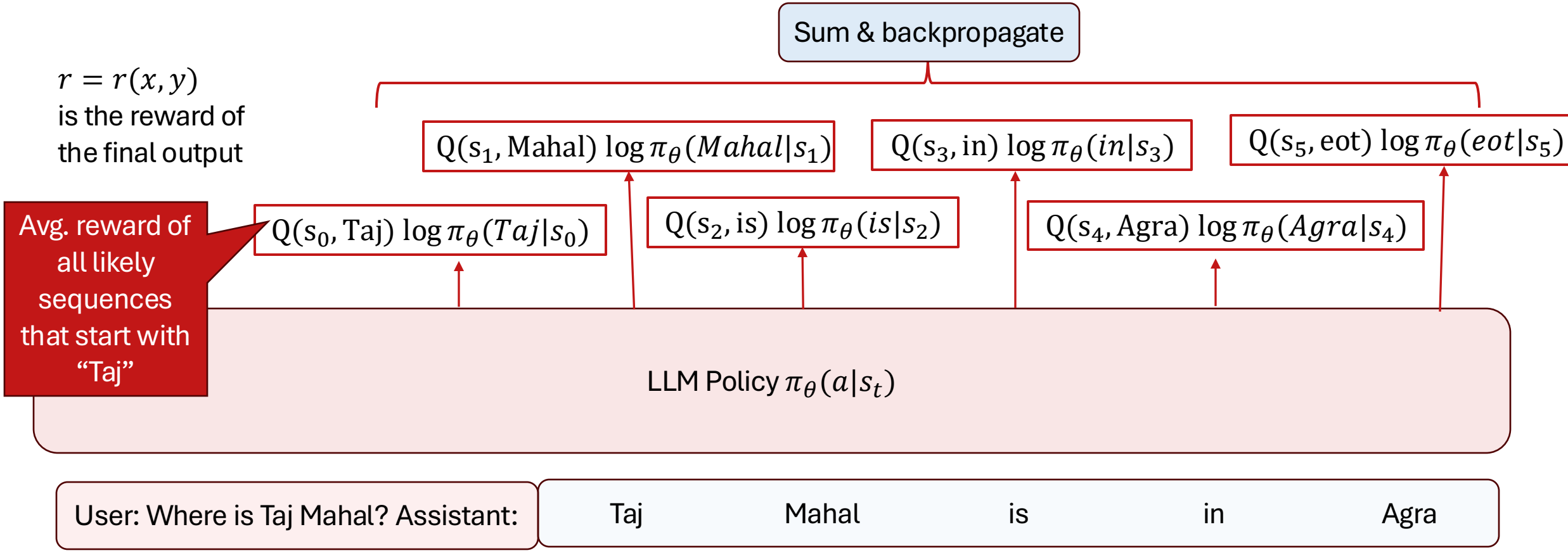
- The reward at token “Taj” depends on the tokens generated in the future
- If the model had generated “Taj Mahal is in Paris”
  - The reward would be negative
  - The probability of generating Taj would be decreased
- If the model had generated “Taj Mahal is in Agra”
  - The reward would be positive
  - The probability of generating Taj would be increased
- This variance in the reward leads to unstable training.
- To reduce variance – take the average reward over all likely sequences (under the policy) that generate “Taj” for the first token.
- This is called the $Q - function$

REINFORCE with Q-Functions



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

REINFORCE with Q-Functions



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

Q-function & Value function

- The Q-function for a state-action pair is the average discounted cumulative reward received at the state after taking taking the specified action.

$$Q(s_t, a_t) = \mathbb{E}_{\pi_\theta(a_{t+1}, a_{t+2}, \ldots a_{t+T} | s_t)} \left[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 \ldots \right]$$

$s_{t+1} = (s_t, a_t)$

γ: discount factor.

- The discount factor γ ensures that immediate rewards get higher weight.
- The Value function of a state is the average discounted cumulative reward received after reaching the state.

$$V(s_t) = \mathbb{E}_{\pi_\theta(a_t, a_{t+1} \ldots a_{t+T} | s_t)} \left[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 \ldots \right]$$

$$Q(s, a) = r(s, a) + \gamma V^*(\delta(s, a))$$

From Q-Function to Advantage Function

- For text generation using language models

$$s_{t+1} = (s_t, a_t)$$

- That is, once you have generated the next token, the next state is determined completely.
- Hence, the Q-function for a state-action pair can be written as

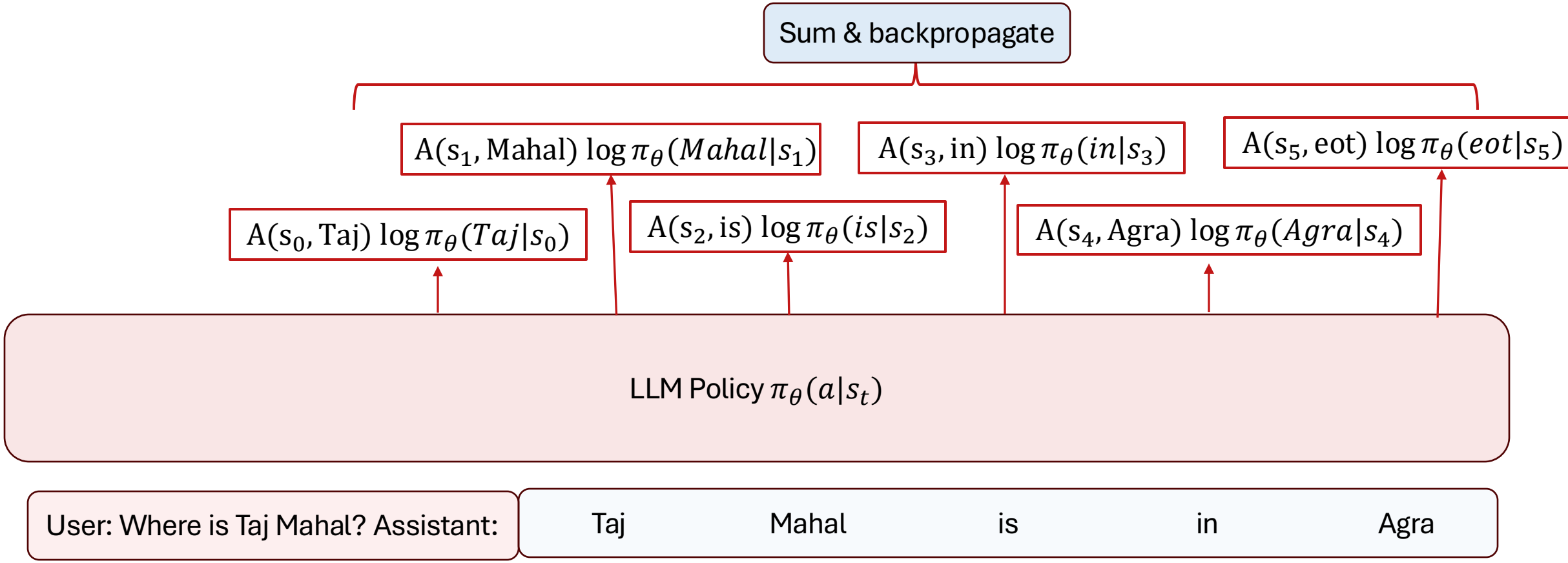
$$Q(s_t, a_t) = r(s_t, a_t) + \gamma\, V(s_{t+1})$$

- To further reduce variance, the advantage function $A(s_t, a_t)$ is used instead of Q-function

$$\begin{aligned} A(s_t, a_t) &= Q(s_t, a_t) - V(s_t) \\ &= r(s_t, a_t) + \gamma\, V(s_{t+1}) - V(s_t) \end{aligned}$$

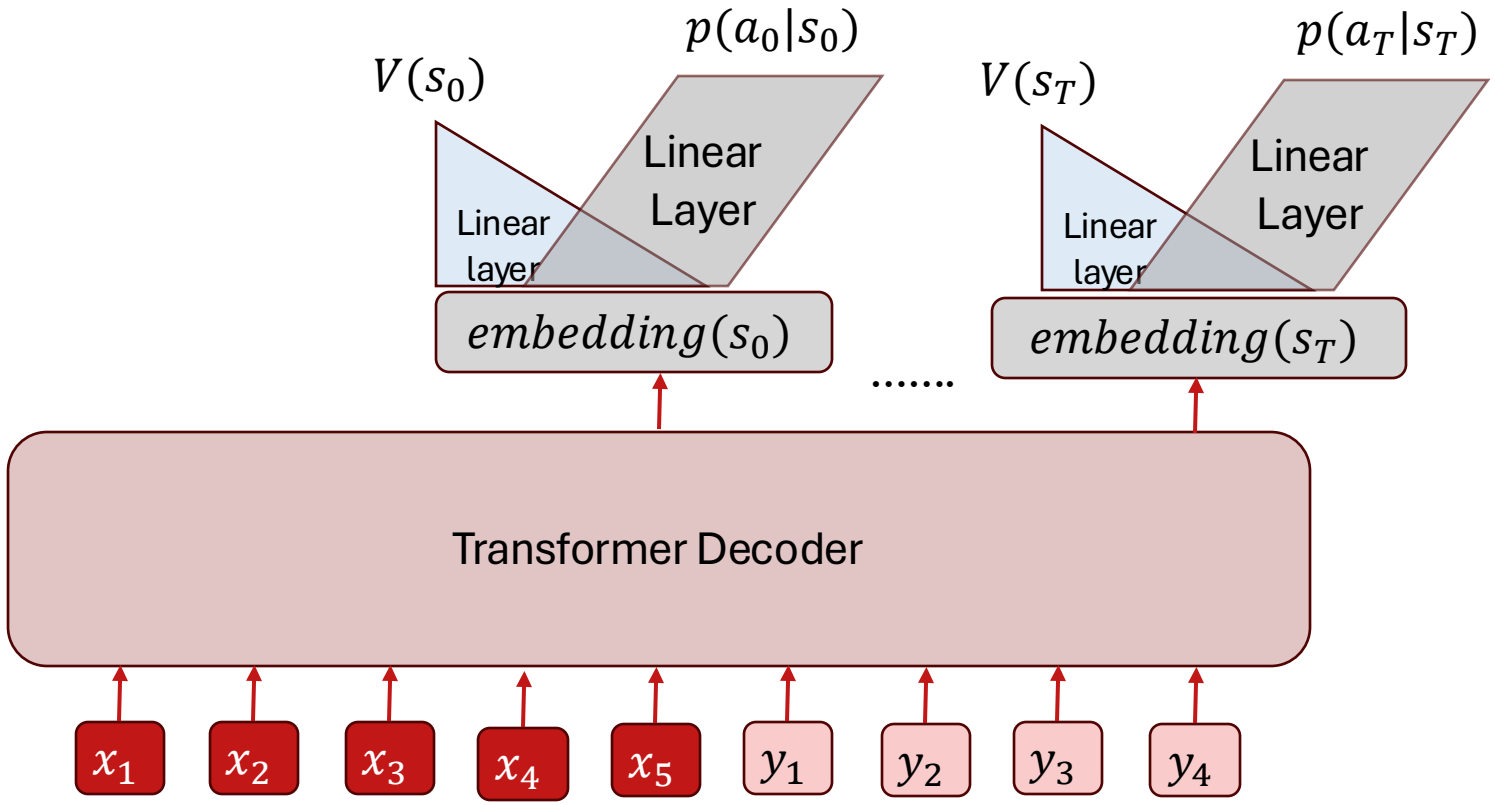
- Intuitively, advantage function captures contribution of the action over an average action at the same state.

REINFORCE with Advantage Functions



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

Implementing the Value Function



Learning the Value function

- Given an input x, sample $y = (a_0, \ldots, a_T)$ from the policy $\pi_\theta(y|x)$
- Compute the cumulative discounted reward for each time-step

$$R_t = r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \ldots$$

Reward-to-go

- Minimize the mean-squared error

$$\min_\phi \sum_{t=0}^{T} \left(V_\phi(s_t) - R_t \right)^2$$

Vanilla Policy Gradient

- Repeat until convergence
  - Sample a batch of prompts B
  - For each prompt, sample one-or more outputs
  - For each output $y = (a_1, \ldots, a_T)$
    - Compute the reward r_t at each token a_t
    - Compute cumulative discounted reward R_t for each token
    - Compute the value & advantage function A_t for each token
  - Apply few gradient updates using REINFORCE with the advantage values computed above
  - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Problems

- Sampling from the policy after every update can be challenging.
- Solution: Sample from an older fixed policy instead

$$\mathbb{E}_{\pi_\theta(y|x)} r_s(x,y) = \sum_{y \in Y} \pi_\theta(y|x)\, r_s(x,y) \times \left(\frac{\pi_{\theta_R}(y|x)}{\pi_{\theta_R}(y|x)} \right)$$

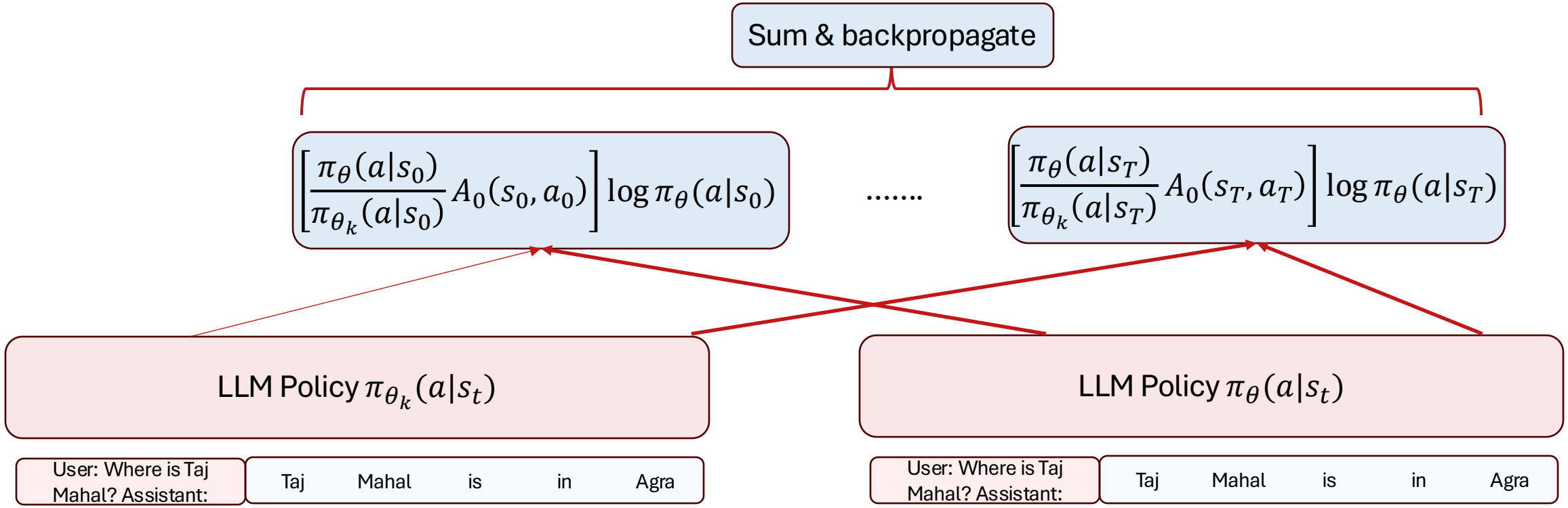
$$= \sum \pi_{\theta_R}(y|x) \left[\frac{\pi_\theta(y|x)}{\pi_{\theta_R}(y|x)} \right] r_s(x,y)$$

$$= \mathbb{E}_{\pi_{\theta_R}(y|x)} \left[\frac{\pi_\theta(y|x)}{\pi_{\theta_R}(y|x)} \right] r_s(x,y)$$

importance weight

REINFORCE with Importance Weights

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



Proximal Policy Optimization

- Keeping the batch of prompts & outputs fixed, how much can we update the policy?
- If we update too much, the importance weights can change drastically.
- PPO-CLIP

$$(1-\epsilon) \leq \frac{\pi_\theta(a_t|s_t)}{\pi_k(a_t|s_t)} \leq (1+\epsilon)$$

- This ensures that the no matter how many updates are done to π_θ, it stays close to π_{θ_t}

PPO-CLIP

$$(1-\epsilon) \leq \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_t}(a_t|s_t)} \leq (1+\epsilon)$$

To achieve above, maximize the following

- When advantage is positive

$$\max_\theta \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, (1+\epsilon)\right) A_t(s_t, a_t)$$

- When advantage is negative

$$\max_\theta \max\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, (1-\epsilon)\right) A_t(s_t, a_t)$$

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

The PPO-CLIP Algorithm

- For $k = 1$ to K
  - Sample a batch of prompts B
  - For each prompt, sample one-or more outputs from $\pi_{\theta_k}(y|x)$
  - For each output $y = (a_1, \ldots, a_T)$
    - Compute the reward r_t at each token a_t
    - Compute cumulative discounted reward R_t for each token
    - Compute the value & advantage function A_t for each token
  - Apply few gradient updates using ~~REINFORCE~~ PPO-CLIP with the advantage values computed above
  - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Policy Gradient/PPO for LLM Alignment

- Collect human preferences (x, y_+, y_-)
- Learn a reward model

$$\phi^* = \underset{\phi}{\text{argmax}} \sum_{(x,y_+,y_-)\in D} \log \sigma(r_\phi(x, y_+) - r_\phi\ (x, y_-))$$

- Train the policy

$$\theta^* = \underset{\theta}{\text{argmax}}\, E_{\pi_\theta(y|x)} r_{\phi^*}(x, y) - \beta . KL(\pi_\theta(y|x)||\pi_{ref}(y|x))$$

- Optionally
  - Also learn the value function

Things to Remember

- The log-derivative trick should be used to compute gradient in REINFORCE
- The log-probability of the tokens should be weighed by the advantage function to reduce variance
- Importance weights should be used to allow sampling from a fixed policy
- The importance weights should be clipped to prevent large gradient updates.