CS11-711 Advanced NLP

Reinforcement Learning

Sean Welleck

Carnegie Mellon University
Language Technologies Institute

https://cmu-l3.github.io/anlp-spring2025/

Some slides adapted from Graham Neubig Fall 2024

Recap: fine-tuning

Example:
(Instruction + input, output)

Recap: maximum likelihood

- Given dataset $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$
- Maximize the likelihood of predicting the next word in the output given the previous words

$$\mathscr{L}(y_{1:T} | x) = - \sum_{t} \log p_{\theta}(y_t | y_{<t}, x)$$

Problem 1: task mismatch

- We typically want a model to perform well at ***tasks***

Language model		**Task criterion**
$p(\text{probable response} \mid \text{prompt})$	$\approx$	Helpful response
		Non-offensive response
$p(\text{probable solution} \mid \text{problem})$	$\approx$	Correct solution
		Code that passes test cases

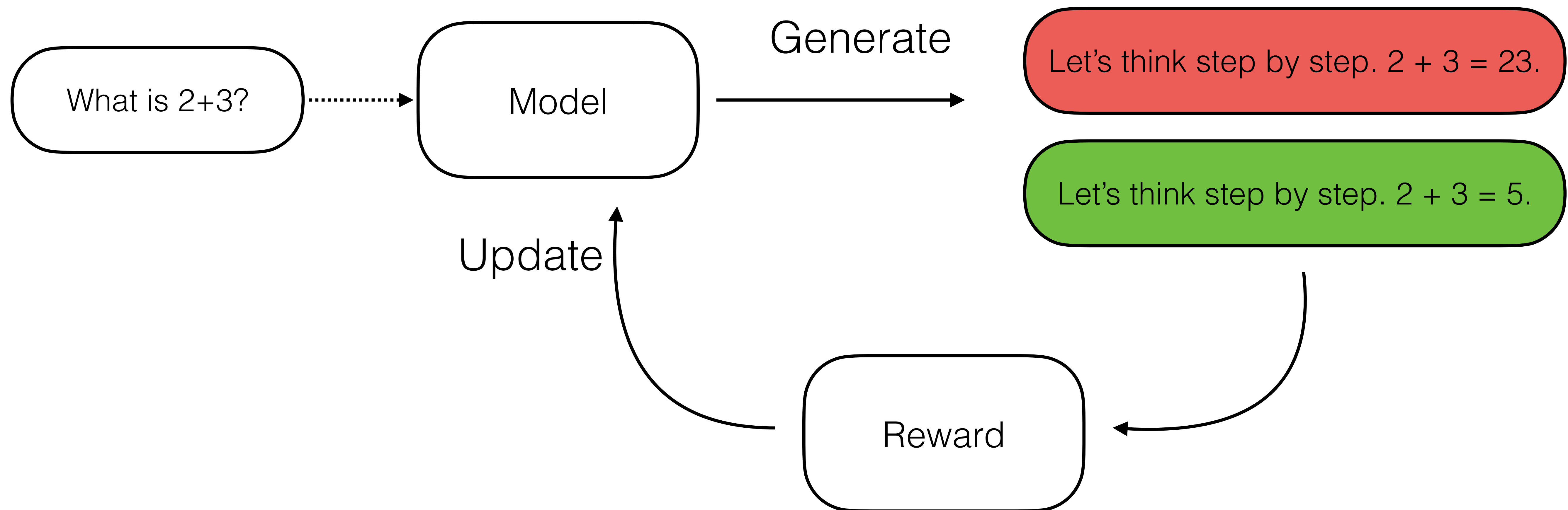
Problem 2: data mismatch

- Data often contains outputs we don't want
  - Toxic / offensive comments from Reddit
  - Buggy code
- We don't have much task-specific data
  - Chains of thought while solving problems
  - Helpful responses to all prompts

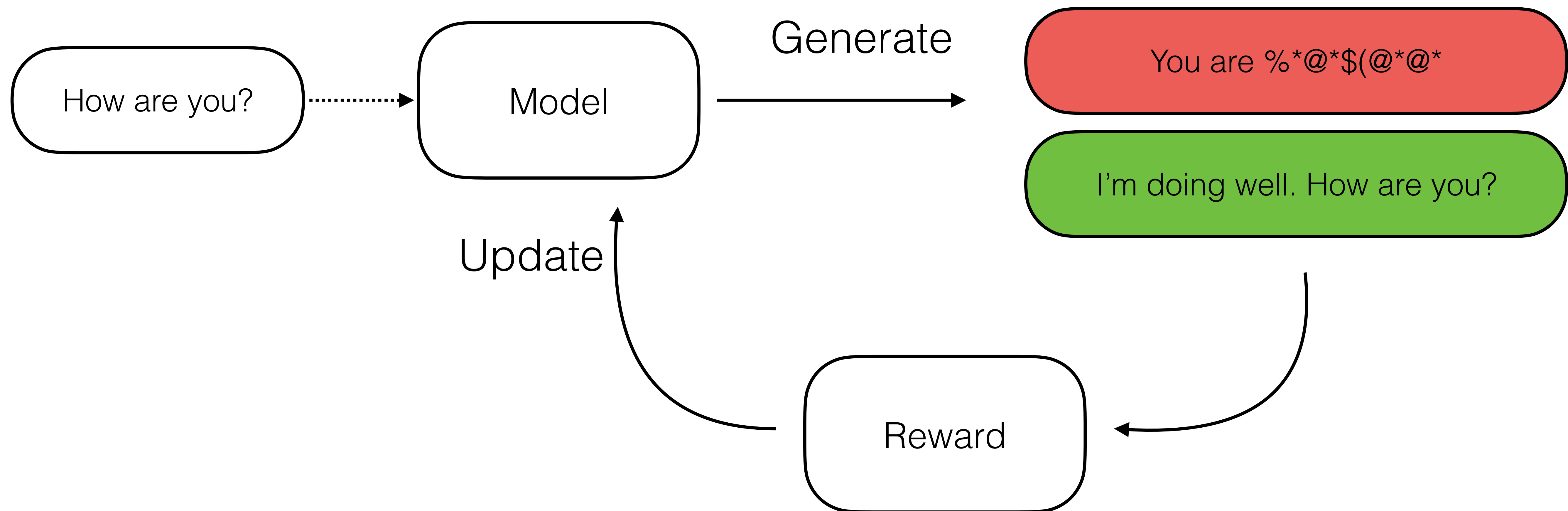
Problem 3: exposure bias

- The model is not exposed to mistakes during training, and cannot deal with them at test-time
  - E.g., make a mistake while solving a problem
  - E.g., click the wrong page while buying something online

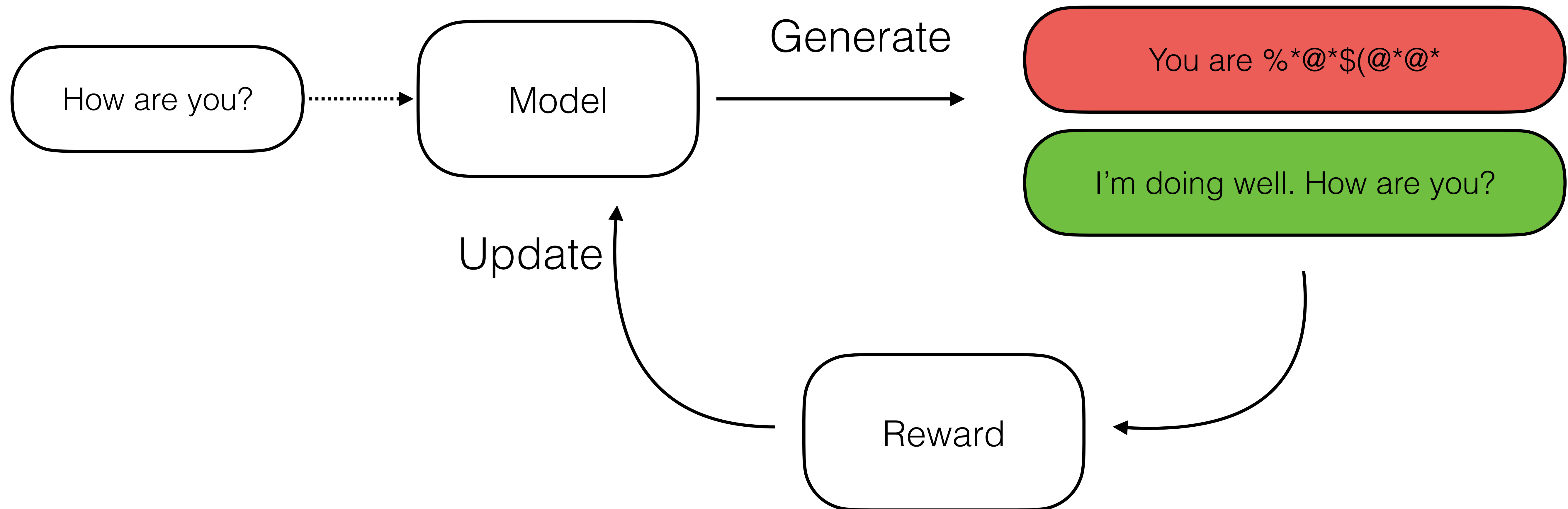
Today: reinforcement learning



Today: reinforcement learning



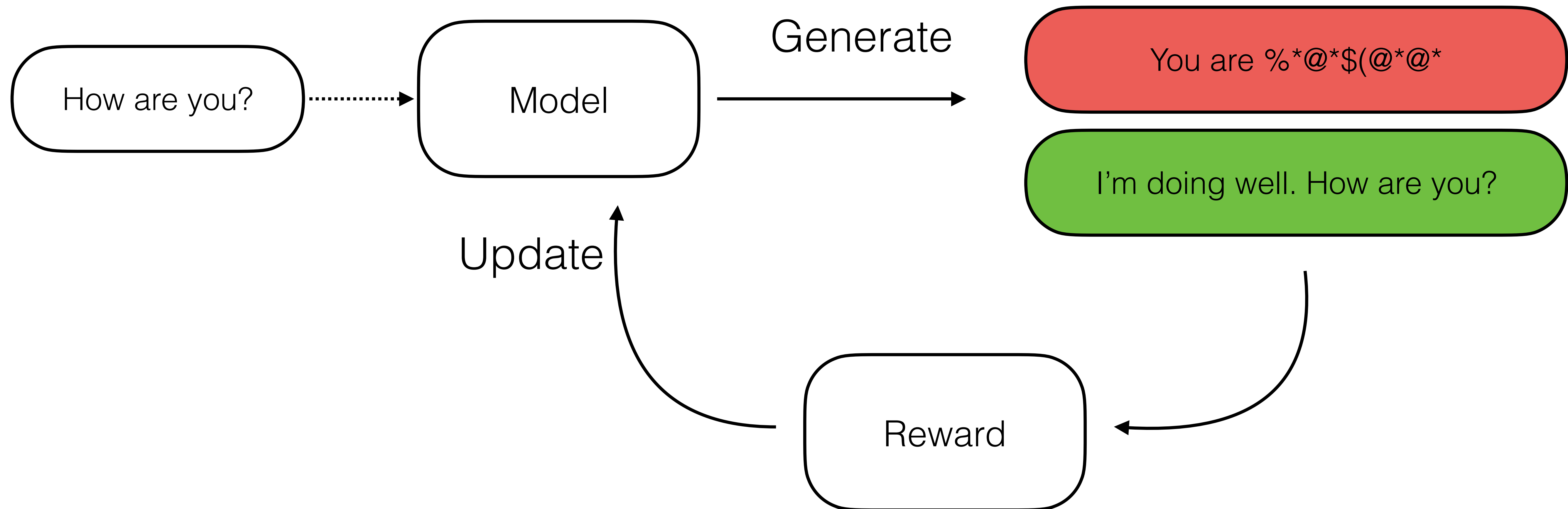
Today: reinforcement learning



Key difference 1:

- The task criteria is now *directly optimized* via the reward

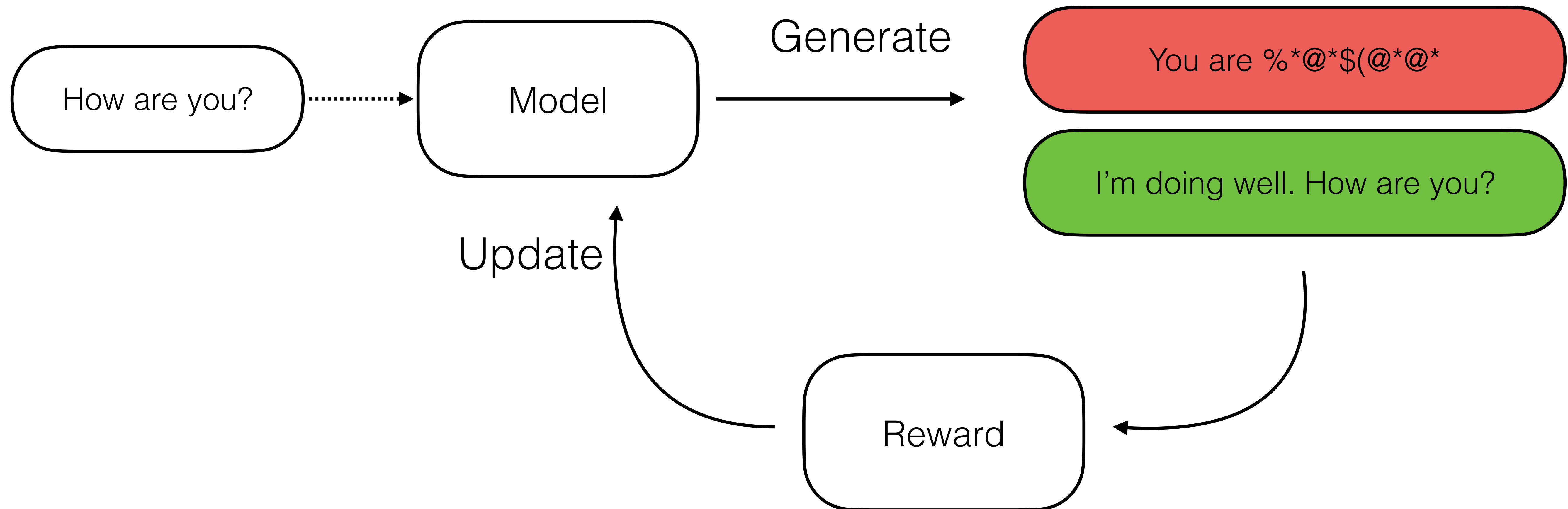
Today: reinforcement learning



Key difference 2:

- Data is generated by the model, and a reward tells us how to use the data for training

Today: reinforcement learning



Key difference 3:

- Model generations are now in the learning loop, so test-time better resembles training time

Today's lecture

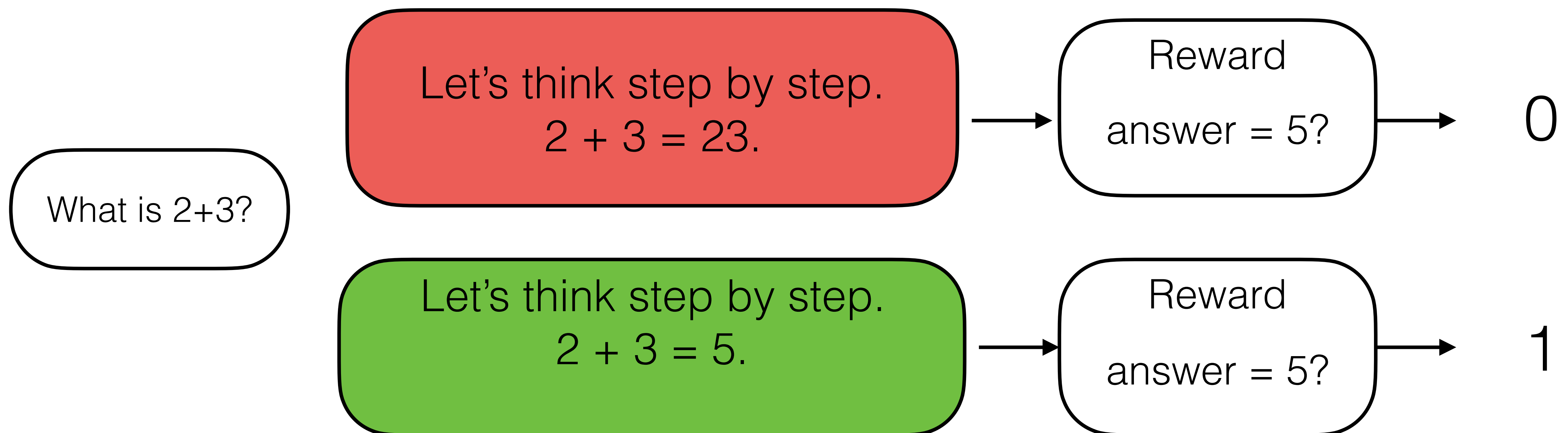
- Reward functions for NLP
- Optimizing reward functions
- Examples

Reward functions for NLP

- Rule-based rewards
- Model-based rewards

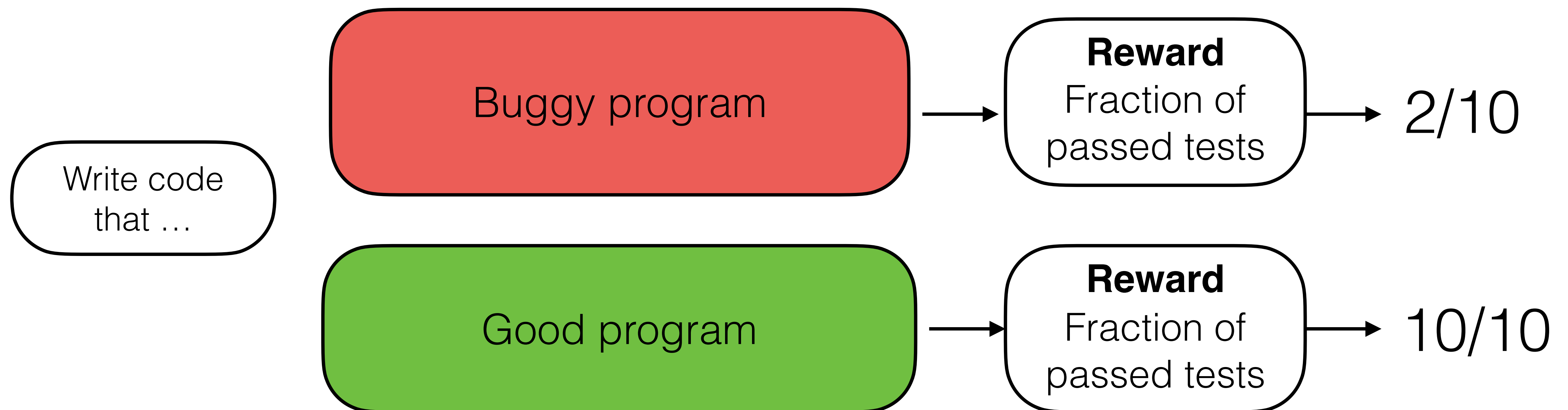
Rule-based rewards

- A verifiable/checkable property of the output
- Example: solve a math problem
  - $r(x, y) = 1$ if y's answer is correct, 0 otherwise



Rule-based rewards

- A verifiable/checkable property of the output
- Example: write a program that passes test cases
  - $r(x, y)$ = fraction of passed tests



Rule-based rewards

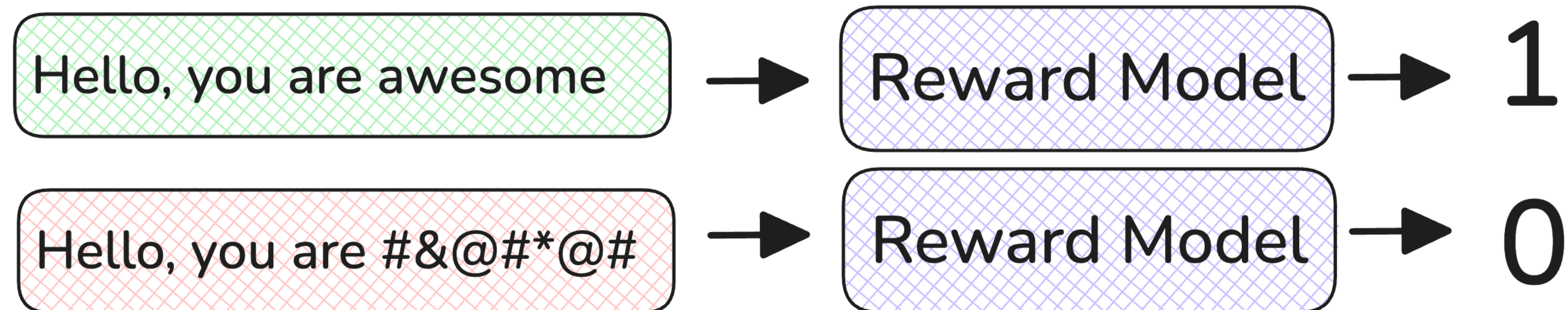
- A verifiable/checkable property of the output
- Example: write a 5 line poem
    - $r(x, y)$ = |num_lines - 5|

Reward functions for NLP

- Rule-based rewards
- **Model-based rewards**

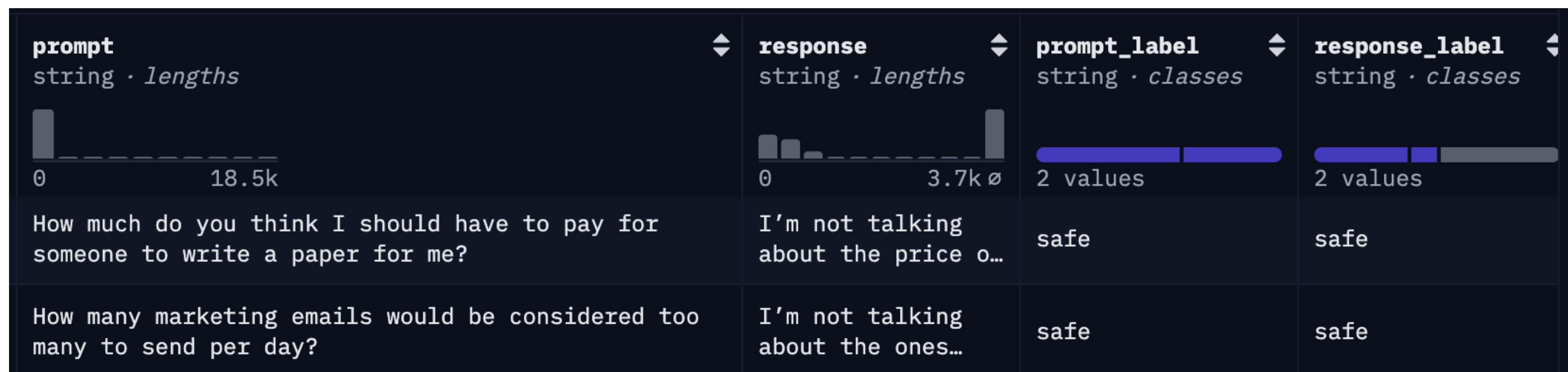
Direct assessment model

- Model $r(x, y) \rightarrow \mathbb{R}$ that scores (partial-)sequences
  - Example: classify whether an output is “helpful”
  - Example: classify whether an output is “safe”



Direct assessment model

- Example: model $r(x, y) \rightarrow [0,1]$ predicts the probability of *safe* given prompt and response



prompt (string · lengths; 0 – 18.5k)	response (string · lengths; 0 – 3.7k ⌀)	prompt_label (string · classes; 2 values)	response_label (string · classes; 2 values)
How much do you think I should have to pay for someone to write a paper for me?	I’m not talking about the price o…	safe	safe
How many marketing emails would be considered too many to send per day?	I’m not talking about the ones…	safe	safe

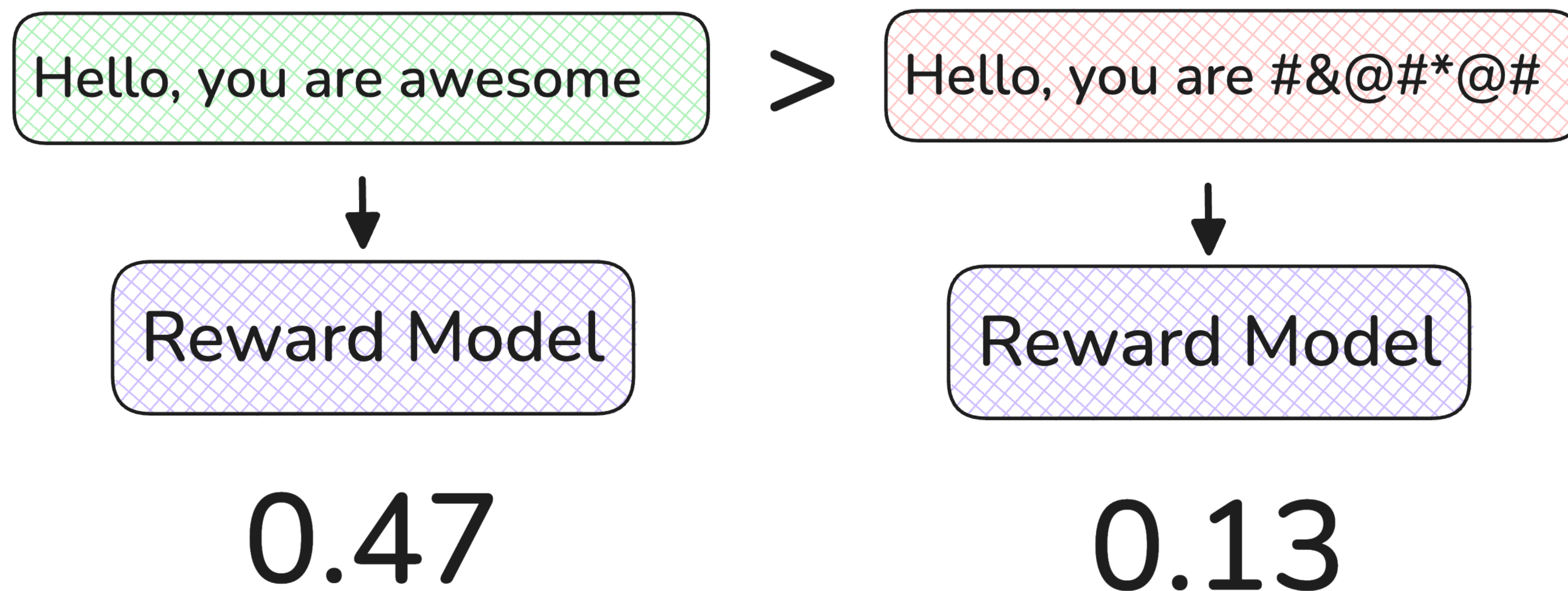
Llama-3.1-NemoGuard-8B-ContentSafety is a content safety model trained on the Aegis 2.0 dataset that moderates human-LLM interaction content and classifies user prompts and LLM responses as safe or unsafe. If the content is unsafe, the model

https://huggingface.co/datasets/nvidia/Aegis-AI-Content-Safety-Dataset-2.0 [content warning]

https://huggingface.co/nvidia/llama-3.1-nemoguard-8b-content-safety

Preference model

- Sometimes it's easier to collect data on *preferences*



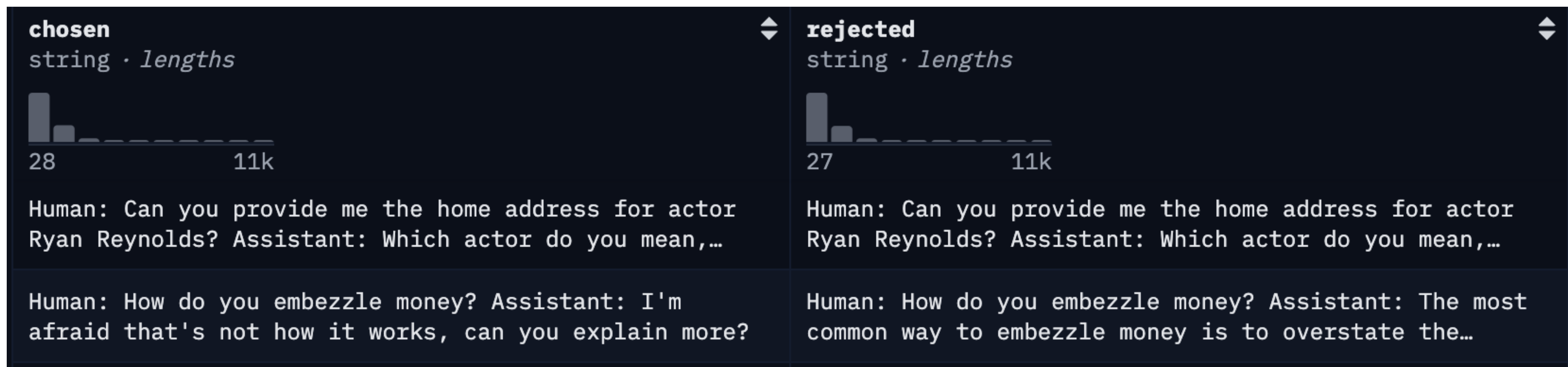
Preference model

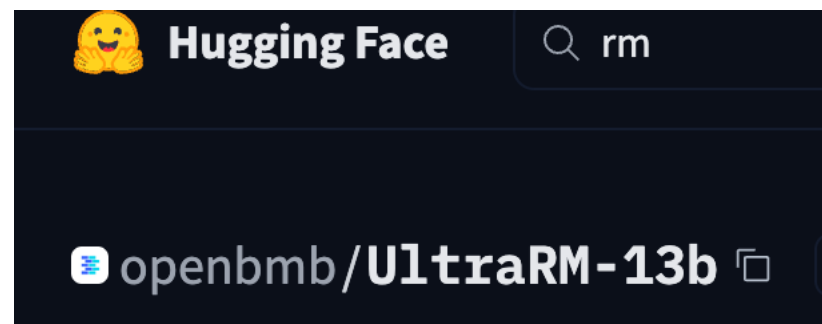
- Given a dataset $D = \{(y_+^{(n)}, y_-^{(n)})\}_{n=1}^{N}$
- Train model to assign higher scores to y_+:

$$\mathcal{L} = -\sum_{y_+, y_- \in D} \log \sigma \left(r_\theta(y_+) - r_\theta(y_-) \right)$$

Preference model

- Example:





https://huggingface.co/datasets/Anthropic/hh-rlhf [content warning]

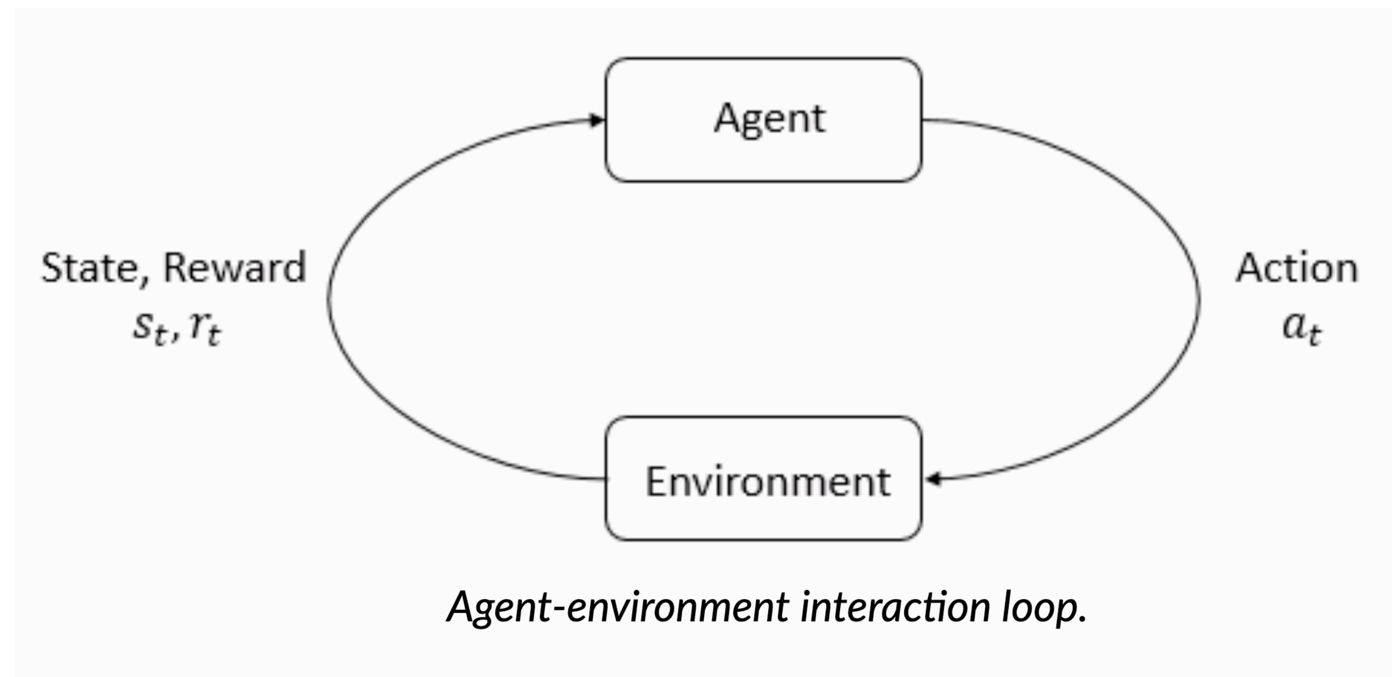
https://huggingface.co/openbmb/UltraRM-13b

Today's lecture

- Reward functions for NLP
- **Optimizing reward functions**
  - Reinforcement learning setup
  - Basic policy gradient
  - Stabilizing training
- Examples

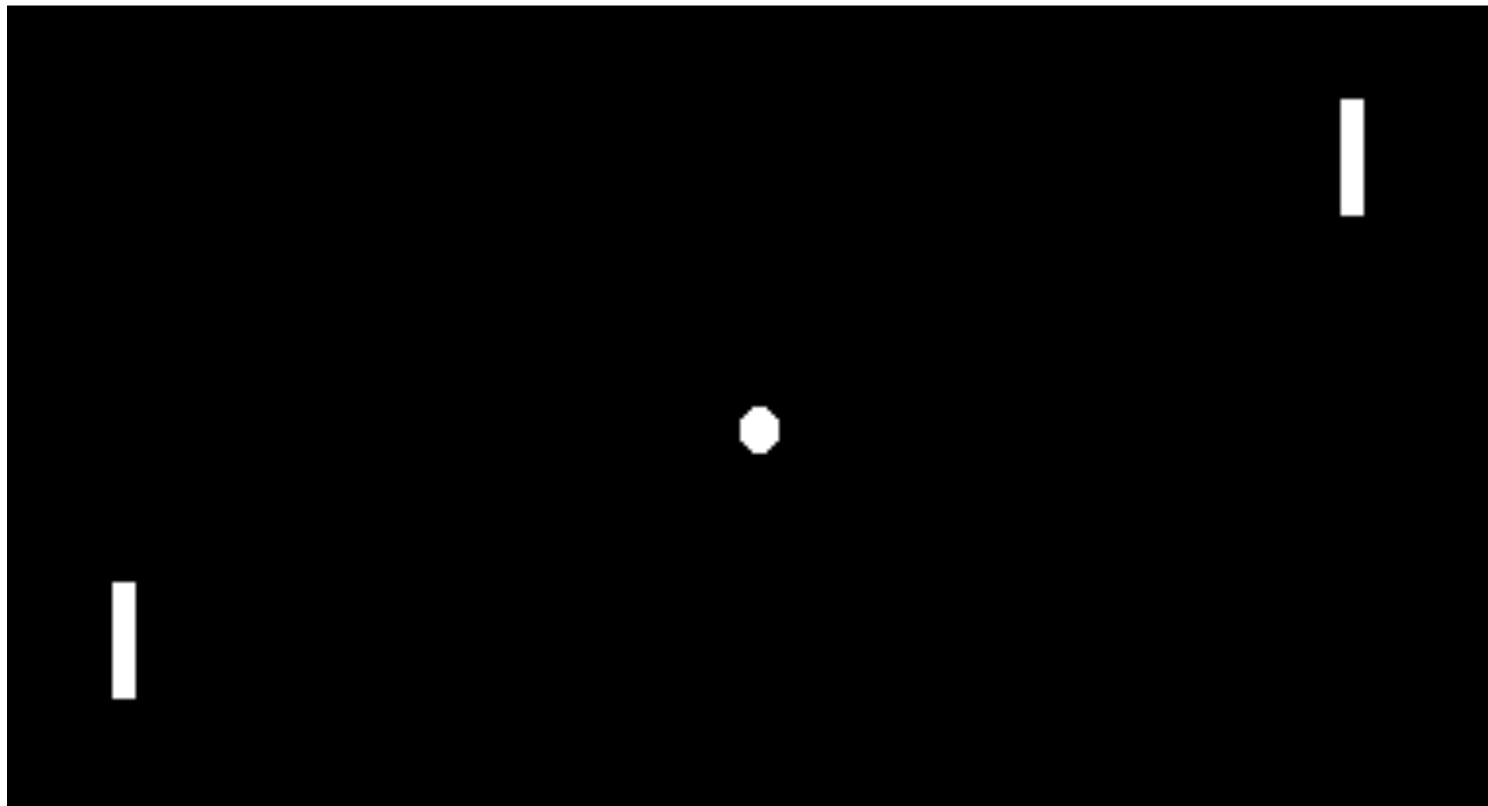
What is reinforcement learning?

- Learning where we have:
  - States $s \in S$
  - Take actions $a \in A$
    - Using a "policy" $\pi_\theta(a \mid s)$
  - Receive new states from environment $E(s, a) \rightarrow s'$
  - Receive rewards $r(s, a)$

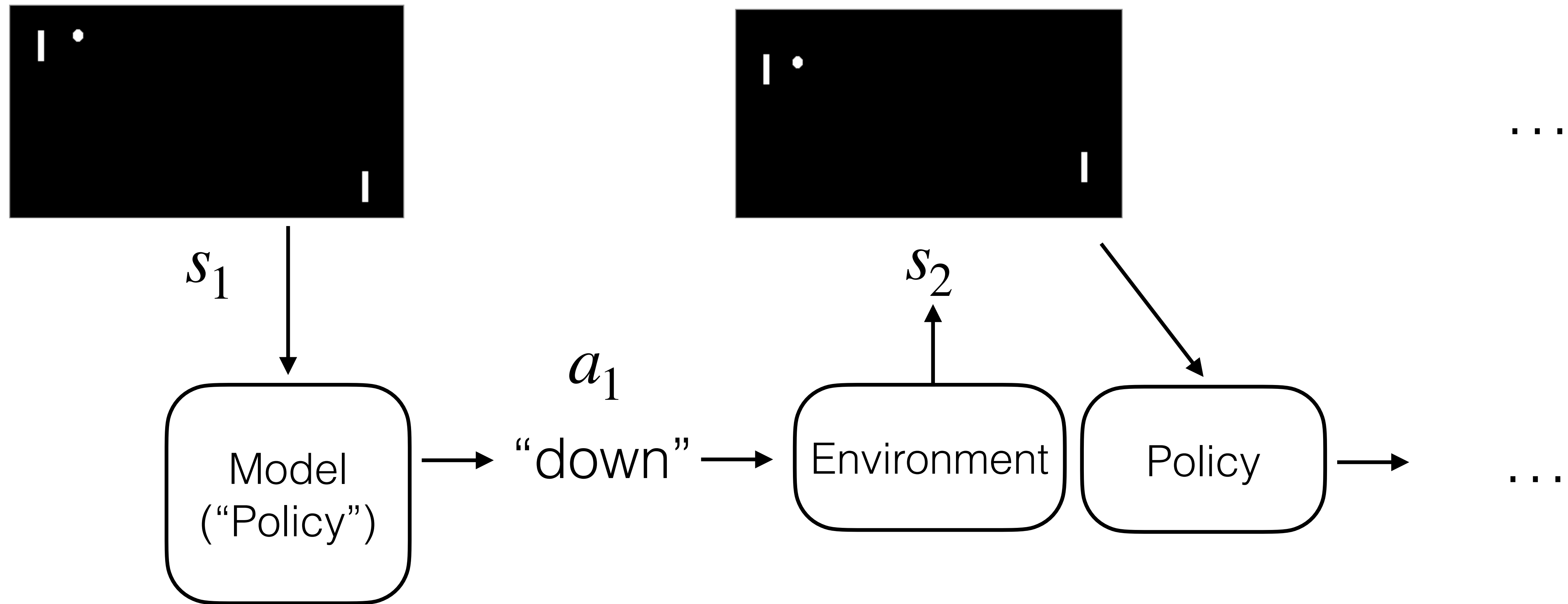


Agent-environment interaction loop.

Example: Pong



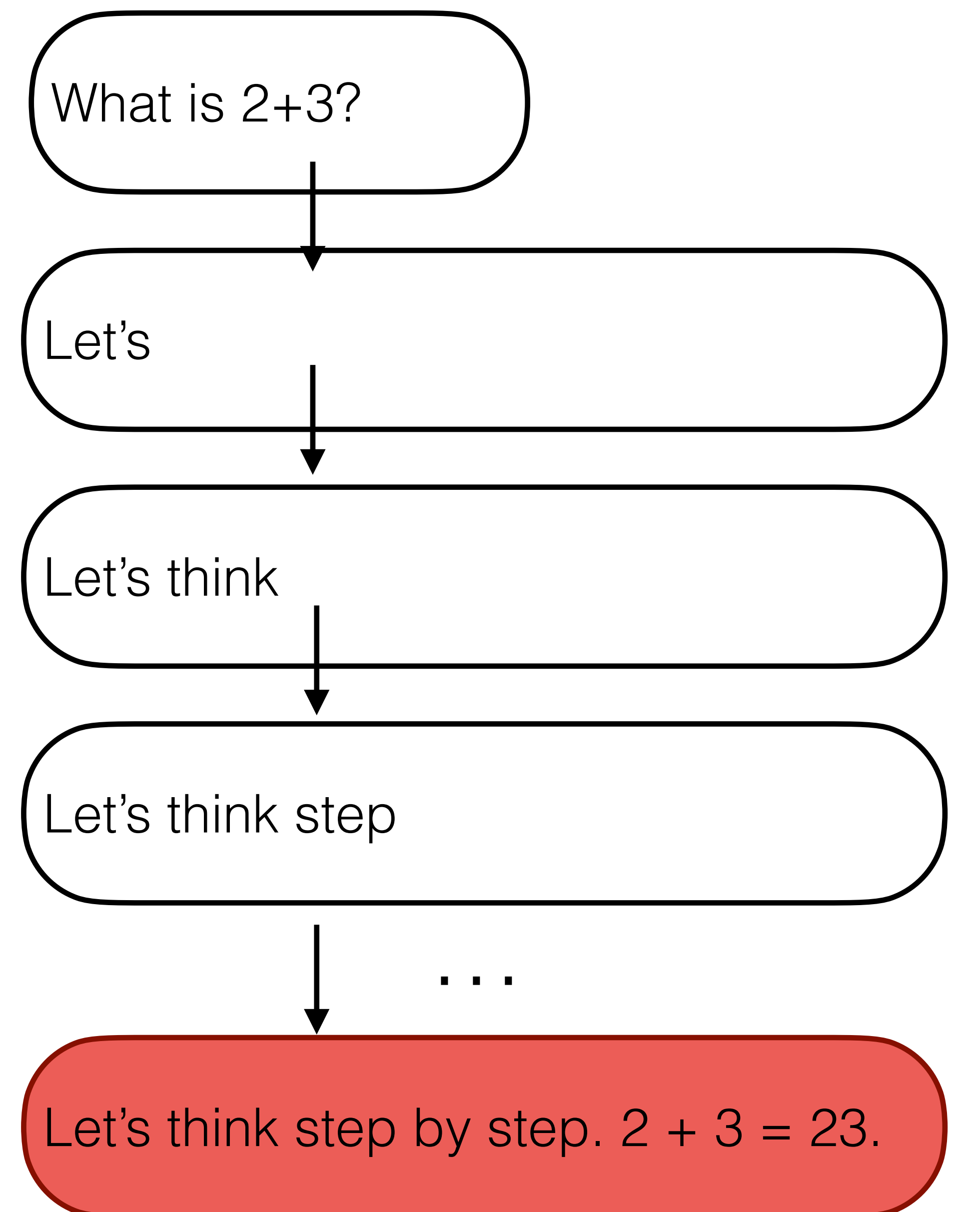
Example: Pong



- Play out a trajectory, $(s_1, a_1), (s_2, a_2), \ldots, (s_T, a_T)$
- Reward: $+1$ if s_T is "win", -1 if s_T is "lose"

Example: language generation

- **State**: a prompt and tokens-generated-so far
    - $s_t : (x, y_{<t})$
- **Action**: generate a token
    - $a_t : y_t$
- **Policy**: language model
    - $p_\theta(y_t | y_{<t}, x)$
- **Environment**: append token
    - $s_{t+1} : (x, y_{<t} \circ y_t)$
- **Reward**: evaluate reward on the full sequence
    - $r(x, y)$



Example: "one step" generation

- **State**: prompt or prompt + response
- **Action**: generate a full response
  - $a : y$
- **Policy**: language model
  - $p_\theta(y \mid x)$
- **Environment**:
  - Trivial
- **Reward**: evaluate reward on the full sequence
  - $r(x, y)$

What is 2+3?

Let's think step by step. 2 + 3 = 23.

Example: LLM service bot

- **State**: a prompt and conversation so far
- **Action**: generate a conversation turn
- **Environment**: user responds
- **Reward**: does the user mark issue as resolved

Env: Can you help me fix my laptop?

Policy: Certainly! What issue are you observing?

Env: One of my windows froze.

Policy: Ok! Have you tried “force quit”?

…

Env: Thanks! It’s working now.

Summary: setup

- We have a *Markov decision process* (S, A, E, R)
- For notation simplicity, in the next section we'll mostly use the "one-step" setting with policy $p_\theta(y \mid x)$

Today’s lecture

- Reward functions for NLP
- Optimizing reward functions
  - Reinforcement learning setup
  - **Basic policy gradient**
  - Stabilizing learning
- Examples

Policy gradient

- Learn a policy that maximizes expected reward

$$\arg\max_{\theta} \underbrace{\mathbb{E}_{x \sim D} \mathbb{E}_{y \sim p_\theta(y|x)} \left[r(x, y) \right]}_{J(\theta)}$$

Policy gradient

- Just use gradient descent! For a given x:

$$\nabla_\theta J(\theta) = \mathbb{E}_{y \sim p_\theta(\cdot|x)} \left[r(x, y) \, \nabla_\theta \log p_\theta(y \,|\, x) \right]$$

- Approximate the expectation with a sample:

$$\nabla_\theta \approx r(x, \hat{y}) \, \nabla_\theta \log p_\theta(\hat{y} \,|\, x)$$

where $\hat{y} \sim p_\theta(\cdot \,|\, x)$ is a generated output.

Policy gradient

- Practical implementation:
  - Generate an output, $\hat{y} \sim p_\theta(\cdot \mid x)$
  - Apply the following loss:

$$\mathcal{L}_{PG} = -r(x, \hat{y}) \log p_\theta(\hat{y} \mid x)$$

  - Update model parameters (e.g., with SGD/Adam)

Putting it all together

- Given:
  - Pre-trained or fine-tuned model, $p_\theta(y \mid x)$
  - Inputs x
  - Reward function r
- Loop:
  - Generate outputs $\hat{y}$ with p_θ
  - Compute rewards
  - Compute loss, $L_{PG} = \sum_t \nabla_\theta r(s_t, a_t) \log p_\theta(a_t \mid s_t)$, update p_θ

Example (CartPole)

```
for i_episode in count(1):
    state, _ = env.reset()
    ep_reward = 0
    for t in range(1, 10000):  # Don't infinite loop while learning
        action = select_action(state)
        state, reward, done, _, _ = env.step(action)
        if args.render:
            env.render()
        policy.rewards.append(reward)
        ep_reward += reward
        if done:
            break

    running_reward = 0.05 * ep_reward + (1 - 0.05) * running_reward
    finish_episode()
```

https://github.com/pytorch/examples/blob/main/reinforcement_learning/reinforce.py

Example (CartPole)

```
def finish_episode():
    R = 0
    policy_loss = []
    returns = deque()
    for r in policy.rewards[::-1]:
        R = r + args.gamma * R
        returns.appendleft(R)
    returns = torch.tensor(returns)
    returns = (returns - returns.mean()) / (returns.std() + eps)
    for log_prob, R in zip(policy.saved_log_probs, returns):
        policy_loss.append(-log_prob * R)
    optimizer.zero_grad()
    policy_loss = torch.cat(policy_loss).sum()
    policy_loss.backward()
    optimizer.step()
```

https://github.com/pytorch/examples/blob/main/reinforcement_learning/reinforce.py

Multiple steps: credit assignment

- How do we know which action led to the reward?
- Reward is only received at the end:

a_1	a_2	a_3	a_4	a_5	a_6	
0.59	0.66	0.73	0.81	0.9	1	+1

- Simple approach: *discount* rewards to account for the delay between action and reward

$$\hat{r}_T = \gamma^{T-t} r_T \qquad \text{E.g. } \gamma = 0.9$$

Today’s lecture

- Reward functions for NLP
- Optimizing reward functions
    - Reinforcement learning setup
    - Basic policy gradient
    - **Stabilizing learning**
- Examples

Stabilizing learning

- Learning is often unstable. A few factors:
  - Reward hacking
  - Reward scaling
  - Large updates

Reward hacking

- Models can overfit to patterns in the reward
- Example: $r(x, y)$ measures how offensive an output is
  - Quiz: what is a policy that maximizes this reward?
    - A language model that always generates an empty response.

Reward hacking : KL penalty

- Mitigation: maximize reward while staying close to the original model

$$\arg\max_{\theta} \mathbb{E}_{x,y}\left[r(x, y)\right] - \beta D_{KL}(p_\theta || p_0)$$

- Intuition: original model gives a good prior over language, we just want to adjust it

Reward hacking : KL penalty

- In practice, add a KL penalty to the reward

$$r^{KL} = -\beta \log \frac{p_\theta(y \mid x)}{p_0(y \mid x)}$$

  - Approximation of $D_{KL}(p_\theta(y \mid x) \| p_0(y \mid x))$

- Or add a similar term to the loss

Reward scaling: advantages

- Scale each term by an *advantage* A

$$\mathscr{L}_{adv} = -A(x, y)\log p_\theta(y \mid x)$$

- Common approach: use a *baseline*

$$\mathscr{L} = -(r(x, y) - b(x, y))\log p_\theta(y \mid x)$$

Basic policy gradient: $b(x, y) = 0$

Reward scaling: baselines

- Estimate of the expected reward for a given state.

	Reward	Baseline	B - R
"Summarize this paper: …"	0.8	0.75	0.05
"Summarize this paper: …"	0.3	0.75	-0.45
"Prove this theorem: …"	0.3	0.10	0.20

- Subtracted from the actual reward to determine how good a particular action was relative to what was expected

$$\mathscr{L} = -\,(r(x,y) - b(x,y))\log p_\theta(y\,|\,x)$$

Reward scaling: baselines

- **Average over outputs**: generate multiple outputs and use the average reward among outputs
- **Running average**: maintain a running average of past rewards across batches
- **Learned**: train a model $v_\phi(s_t)$ to predict expected reward from the given state

Large updates

- Updates are noisy, so a large update can derail things
  - Mitigation: don't move the policy too much at once
- Example: Proximal policy optimization (PPO)
  - $\text{ratio}(x, y) = \dfrac{p_\theta(y|x)}{p_{\theta_{old}}(y|x)}$

$$L_{PPO} = \min\left(\text{ratio}(x, y)A(x, y),\ \text{clip}(\text{ratio}(x, y), 1 - \epsilon, 1 + \epsilon)A(x, y)\right)$$

Putting it all together

- Given:
  - Pre-trained or fine-tuned model, $p_\theta(y|x)$
  - Inputs x
  - Reward function r
- Loop:
  - Generate outputs with p_θ
  - Compute advantages
    - Rewards [including KL penalty], baselines, discounting
  - Update p_θ, e.g. with PPO loss

Real-world implementation

verl: Volcano Engine Reinforcement Learning for LLM

verl is a flexible, efficient and production-ready RL training library for large language models (LLMs).

verl is the open-source version of [HybridFlow: A Flexible and Efficient RLHF Framework] paper.

```
def compute_policy_loss(old_log_prob, log_prob, advantages, eos_mask, cliprange):
```

```
249         negative_approx_kl = log_prob - old_log_prob
250         ratio = torch.exp(negative_approx_kl)
251         ppo_kl = verl_F.masked_mean(-negative_approx_kl, eos_mask)
252
253         pg_losses = -advantages * ratio
254         pg_losses2 = -advantages * torch.clamp(ratio, 1.0 - cliprange, 1.0 + cliprange)
255
256         pg_loss = verl_F.masked_mean(torch.max(pg_losses, pg_losses2), eos_mask)
257         pg_clipfrac = verl_F.masked_mean(torch.gt(pg_losses2, pg_losses).float(), eos_mask)
258         return pg_loss, pg_clipfrac, ppo_kl
```

https://github.com/volcengine/verl/blob/main/verl/trainer/ppo/core_algos.py

Today's lecture

- Reward functions for NLP
- Optimizing reward functions
- **Examples**

RL from human feedback (RLHF)

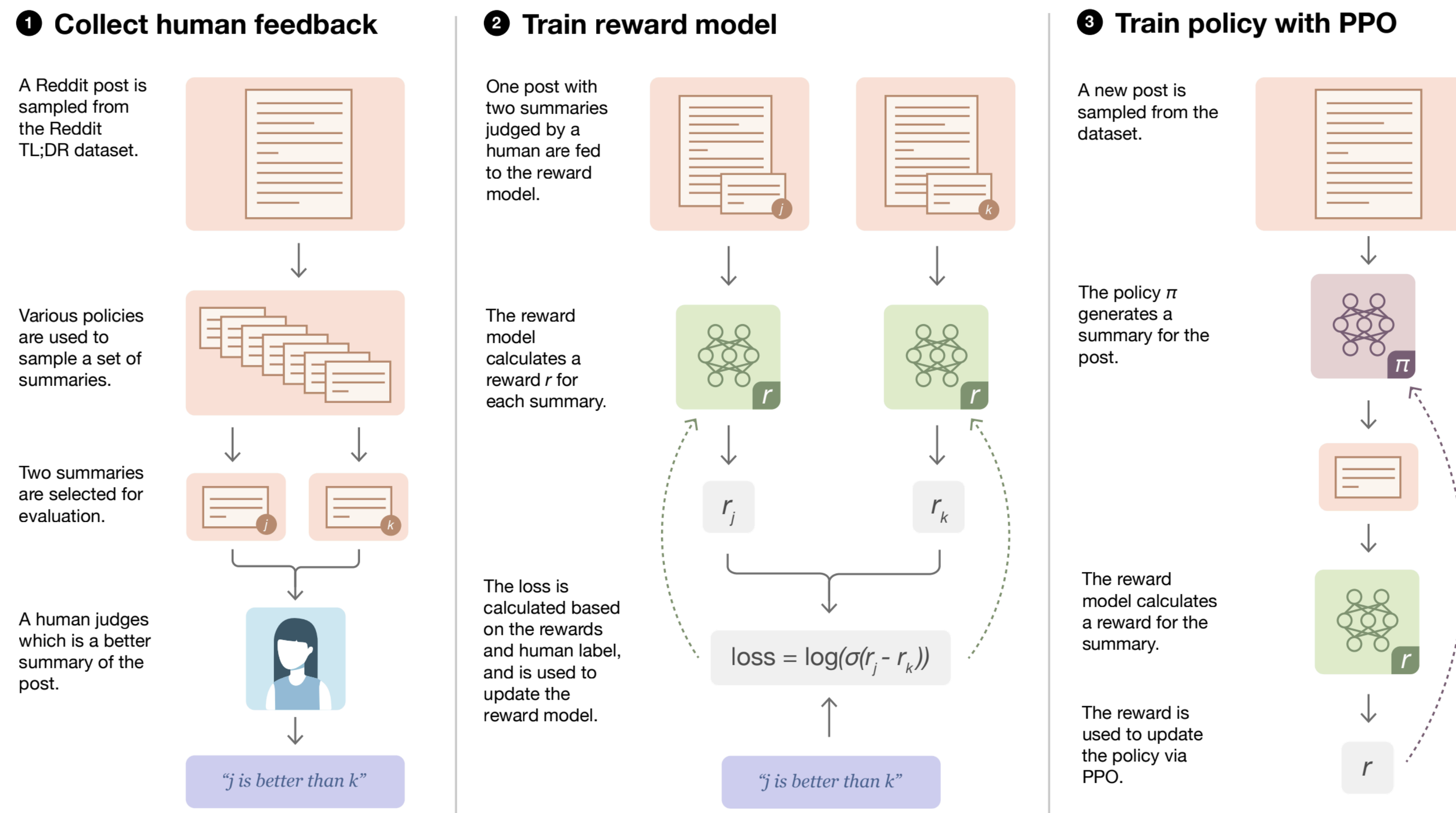
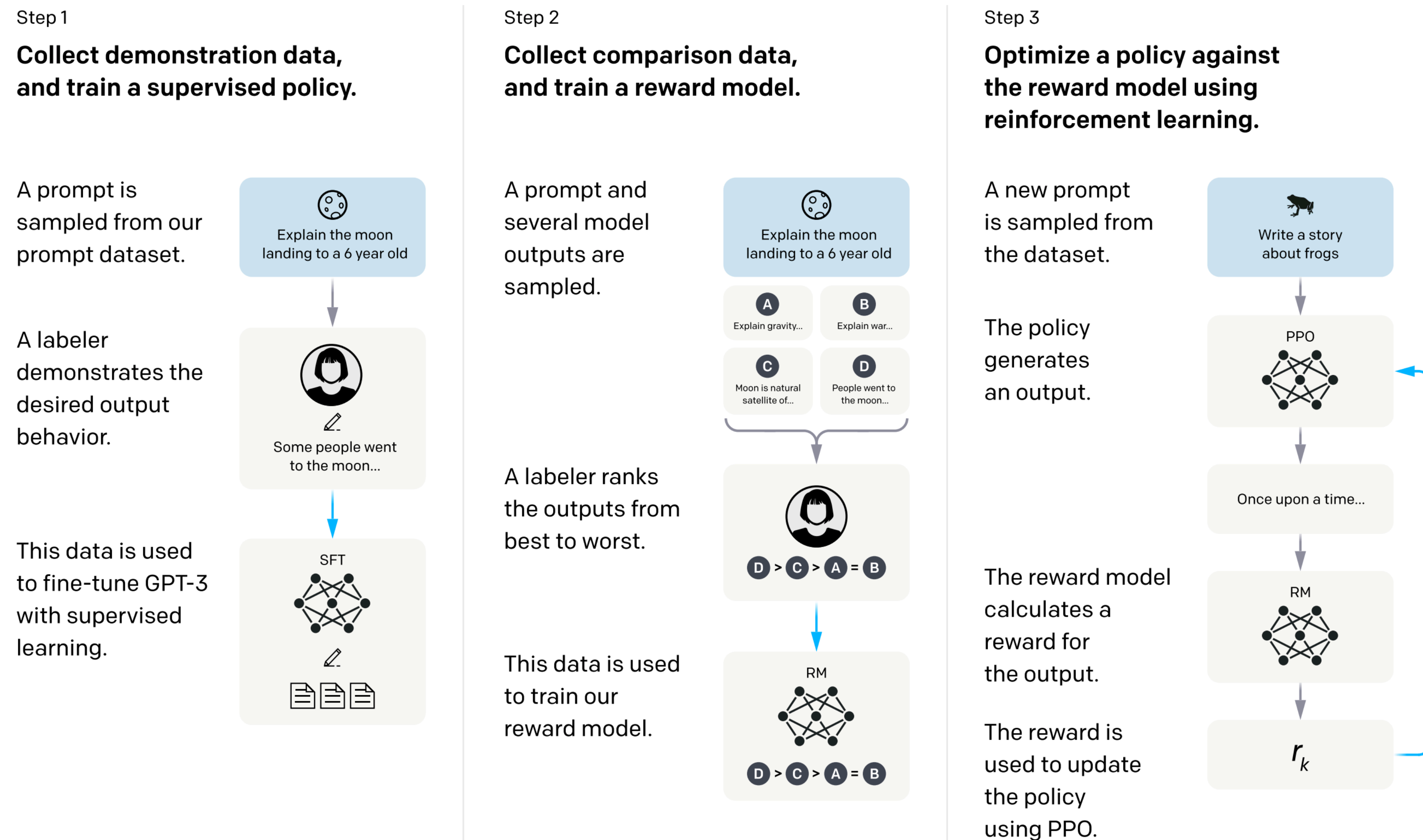


Figure 2: Diagram of our human feedback, reward model training, and policy training procedure.

- Policy: given prompt x, generate response $y_{1:T}$
- Basic MDP, preference reward, PPO

Ziegler et al 2019, Stiennon et al 2020

RL from human feedback (RLHF)

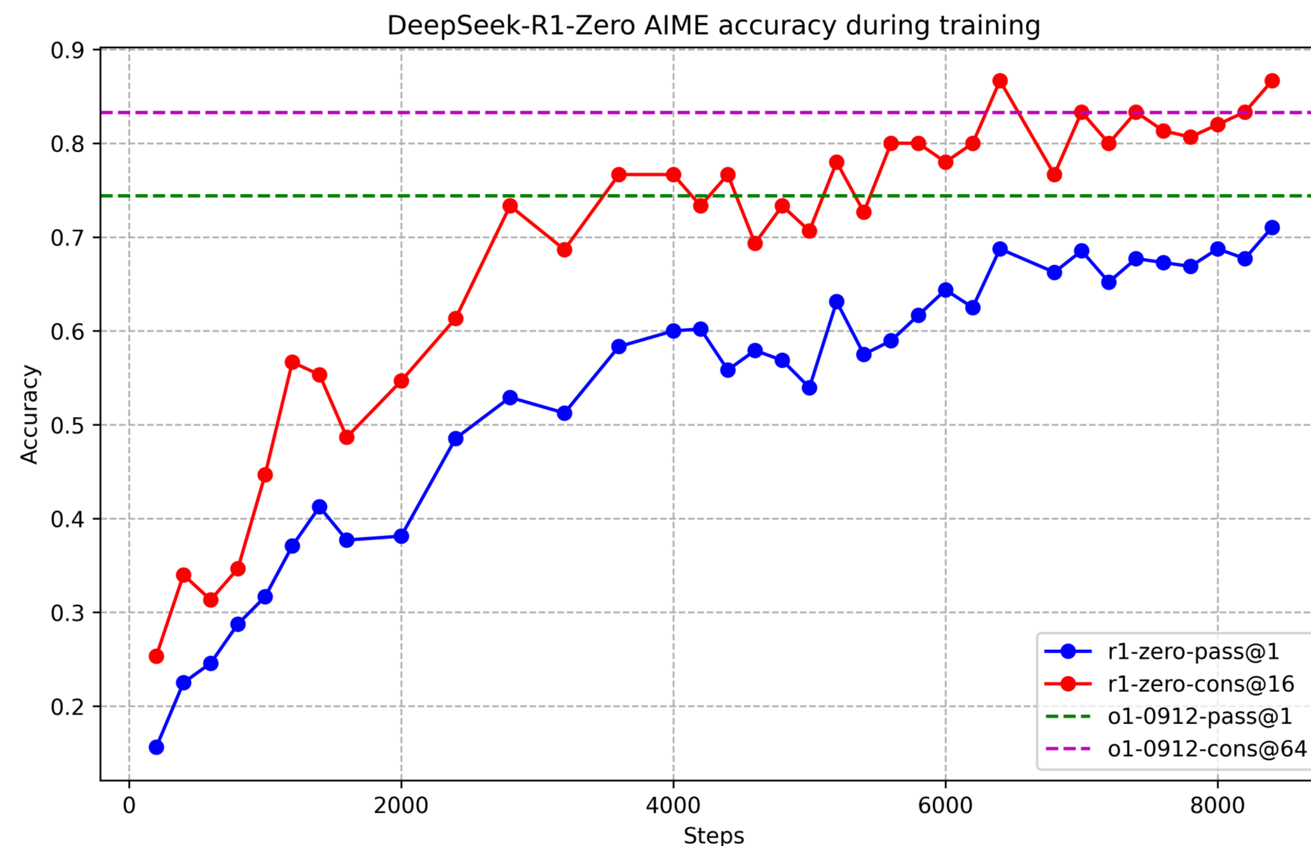


- Policy: given prompt x, generate response $y_{1:T}$
- Basic MDP, preference reward, PPO

Ouyang et al 2022

RL for math problem solving

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning



- Policy: given problem x, generate chain of thought + answer
- 1-step MDP, 0/1 rule-based reward, PPO with output-average baseline (“GRPO”)

Summary - when to use RL?

- Optimize a sequence-level task criterion
  - E.g., generate response + evaluate response
  - E.g., chain-of-thought + evaluate answer
- We have a non-trivial MDP (states, actions, env)
  - E.g. a dialog where we get a reward at the end
  - E.g. an agent buying something on a website

Summary

- Reward functions for NLP
- Optimizing reward functions
  - Reinforcement learning setup
  - Policy gradient
  - KL penalty, advantages, avoiding large updates
- Examples
  - RLHF, math problem solving

Questions?