CS11-711 Advanced NLP

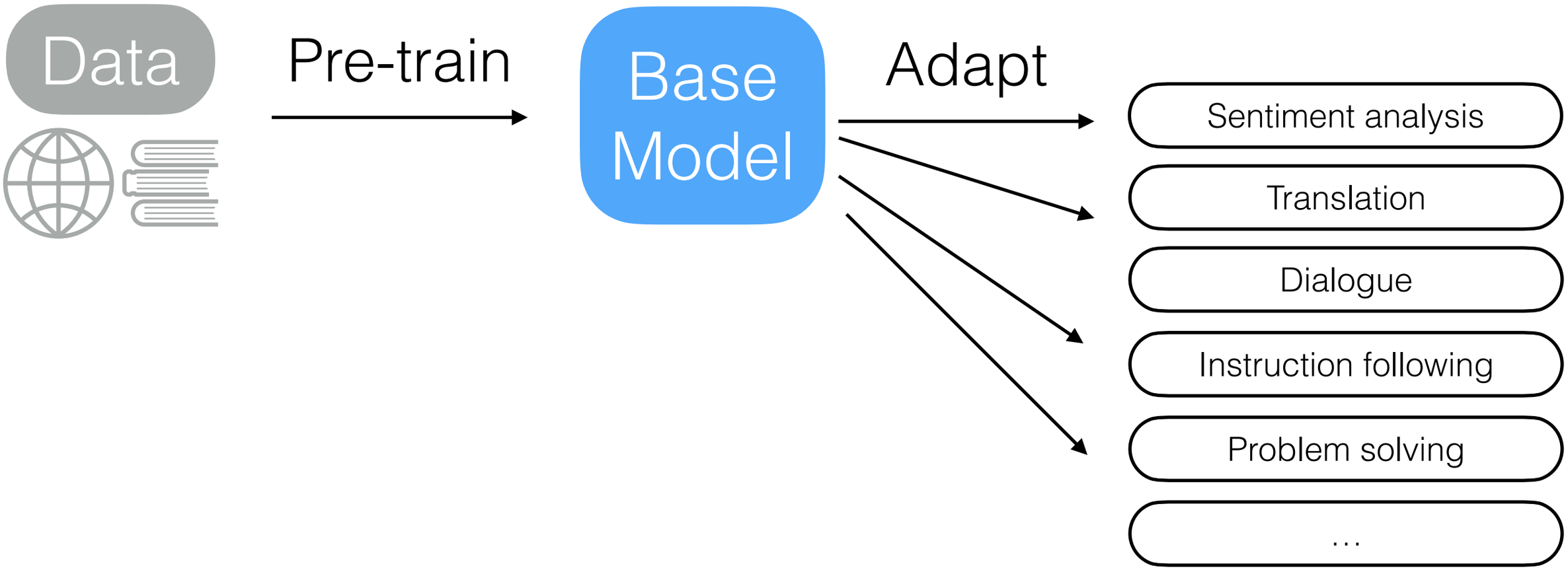
# Advanced Post-Training

Sean Welleck

Carnegie Mellon University
Language Technologies Institute

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

# Recap: pre-training



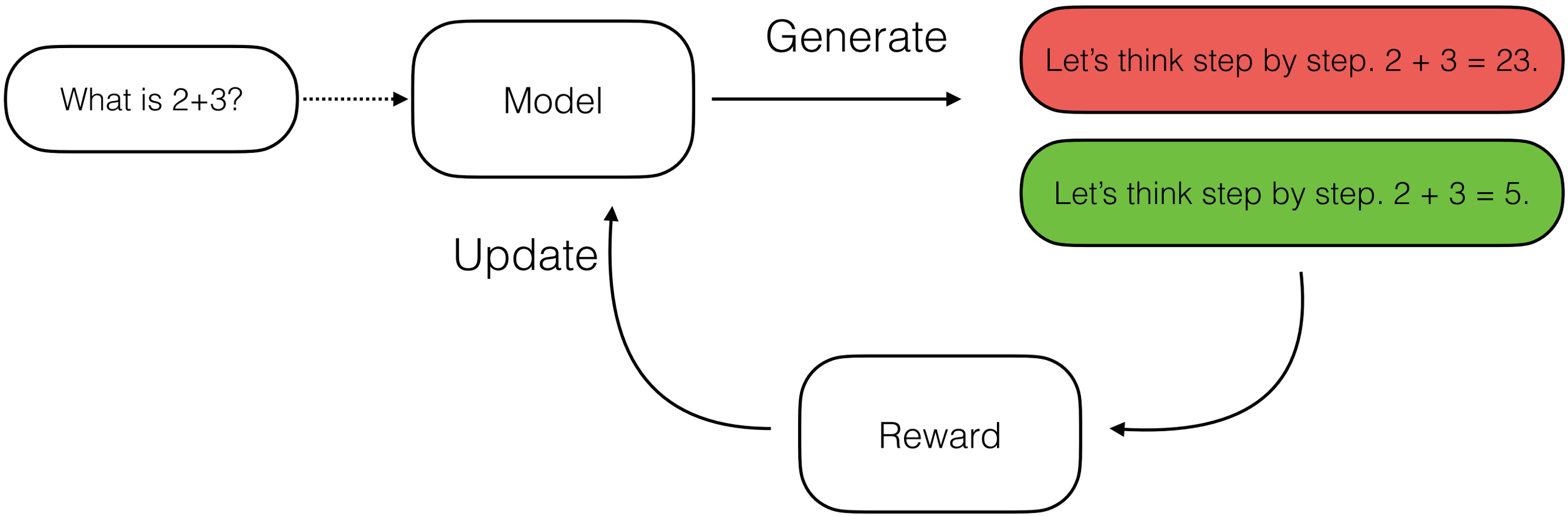
# Post-training

- **Post-training**: keep training a base model
  - Fine-tuning (e.g., instruction tuning)
  - Reinforcement learning
  - Other learning algorithms
- Goal: make the model better for downstream use
  - As a chat model, as a problem solving model, …

# Today's lecture

- **Language model reinforcement learning pipeline**
  - Example: RLHF
    - Supervised fine-tuning
    - Reward modeling
    - Reinforcement learning
- **Direct preference optimization**

# Recap: reinforcement learning



# Key idea: keep training using feedback

- **Example feedback:** reward function
- **Example algorithm**: policy gradient

# Preference Feedback

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

# RL from Human Feedback (RLHF)

1. **Supervised fine-tuning (SFT)**: Fine-tune a language model using a dataset $D_{SFT} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$.

   - Example: $x$ is a prompt, $y$ is a human-written response

2. **Reward modeling**: Train a reward model $r_\phi(x, y)$ using preference data $D_{Pref} = \{(x, y_+^{(n)}, y_-^{(n)})\}_{n=1}^{N'}$

   - Example: $x$ is a prompt, $y$ is a model-generated response

3. **Reinforcement learning (RL)**: Further fine-tune the language model from step 1 (call it $p_0$) using a reinforcement learning algorithm:

$$p_\theta = RL(p_0, r_\phi, \{x^{(n)}\}_{n=1}^{N''})$$

   - Example: $x$ is a prompt, PPO is the RL algorithm

# RL from Human Feedback (RLHF)

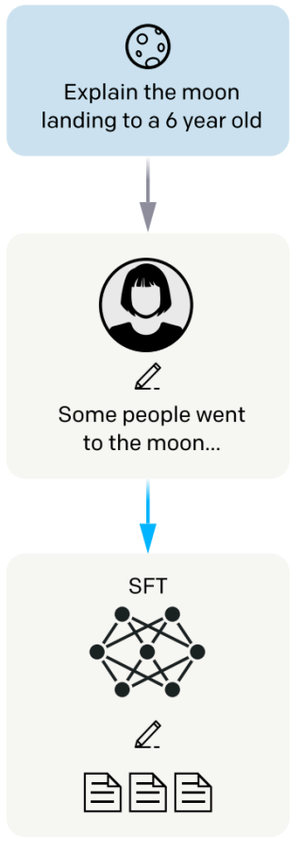
Step 1

**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.



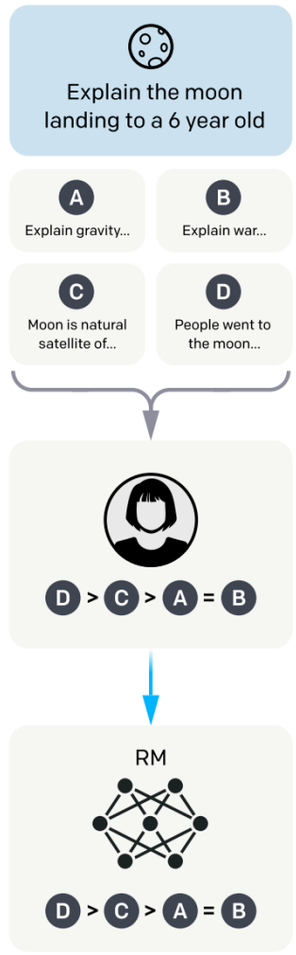

Step 2

**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

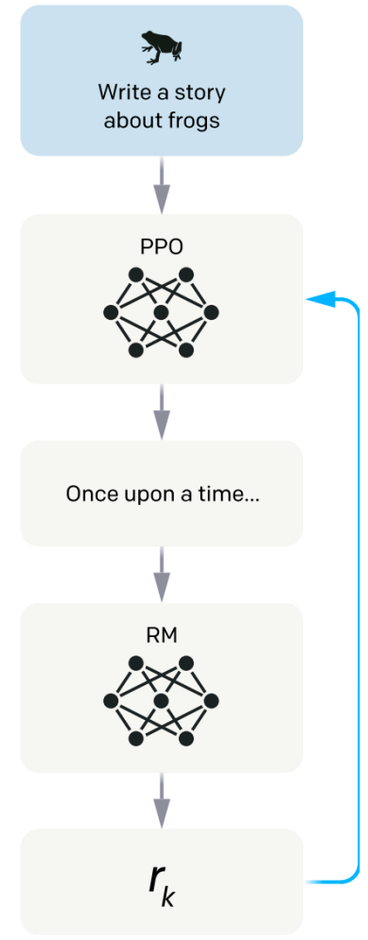
**Optimize a policy against the reward model using reinforcement learning.**

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.




Ouyang et al 2022

# Step 1: SFT

1. **Supervised fine-tuning (SFT)**:
   Fine-tune a language model using a dataset $D_{SFT} = \{(x^{(n)}, y^{(n)})\}_{n=1}^{N}$.

- Example data:
  - Alpaca [Taori et al 2023]: 52,000 model-generated (prompt, response) examples
  - Any technique from the instruction-tuning lecture! (*see Lecture 9)*

Example Generated task

*Instruction: Brainstorm creative ideas for designing a conference room.*

*Output:*
*... incorporating flexible components, such as moveable walls and furniture ...*

# Large-scale example (AI2 Tulu 3)

- 900k prompts:
  - 57%: open-source data
  - 43%: generate additional prompts for instruction following, math, and code. Generate responses using proprietary models
- Skill-specific mixtures: keep the mixtures that led to the best performance on individual skills

| Category | Prompt Dataset | Count |
|---|---|---|
| General | OpenAssistant Guanaco[1,2] | 7,132 |
| | **TÜLU 3 Hardcoded** | 24 |
| | No Robots | 9,500 |
| | WildChat GPT-4[↓] | 241307 |
| | UltraFeedback | |
| Knowledge | FLAN v2[1,2,↓] | 89,982 |
| Recall | SciRIFF[↓] | 35, 357 |
| | TableGPT[↓] | 5,000 |
| Math | **TÜLU 3 Persona MATH** | 149,960 |
| & Reasoning | **TÜLU 3 Persona GSM** | 49,980 |
| | **TÜLU 3 Persona Algebra** | 20,000 |
| | OpenMathInstruct 2[↓] | 50,000 |
| | NuminaMath-TIR[α] | 64,312 |
| Coding | **TÜLU 3 Persona Python** | 34,999 |
| | Evol CodeAlpaca[α,2] | 107,276 |
| Safety & | **TÜLU 3 CoCoNot** | 10,983 |
| Non-Compliance | **TÜLU 3 WildJailbreak[α,↓]** | 50,000 |
| | **TÜLU 3 WildGuardMix[α,↓]** | 50,000 |
| Multilingual | Aya[↓] | 202,285 |
| Precise IF | **TÜLU 3 Persona IF** | 29,980 |
| | Daring Anteater | |
| *Total* | | 939,344 |

# Step 2: Reward modeling

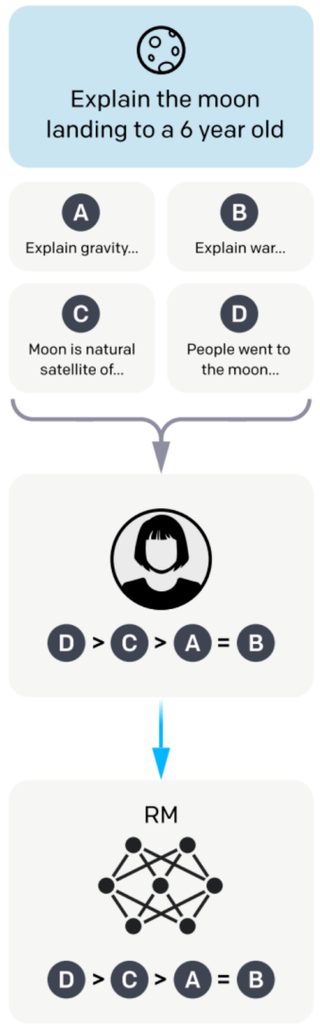
2. **Reward modeling**: Train a reward model $r_\phi(x, y)$ using preference data $D_{Pref} = \{(x, y_+^{(n)}, y_-^{(n)})\}_{n=1}^{N'}$

- Prompts $x$: re-use SFT dataset prompts and/or introduce new ones
    - Example: *AlpacaFarm* [Dubois et al 2024]: reserved out 10k of the 52k Alpaca data for generating $D_{pref}$
- Responses $y$: generate with SFT model or other models
- **Need: (i) preference ratings, (ii) method to train the reward model**

A prompt and several model outputs are sampled.

A labeler ranks the outputs from best to worst.

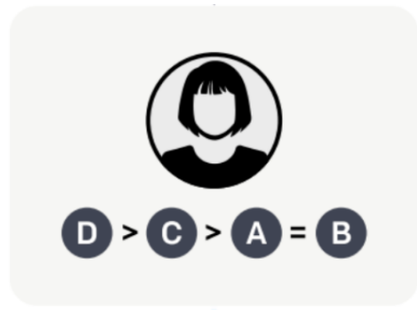
This data is used to train our reward model.



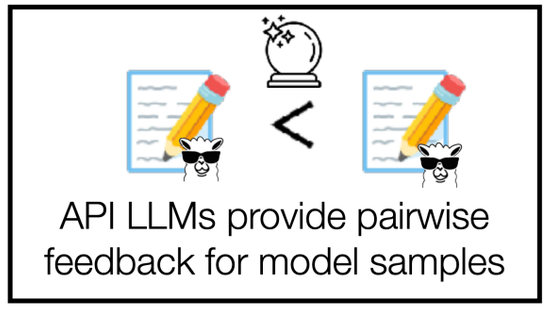
# Gathering preference ratings

- Given $x, y_1, y_2$, determine which response is better (or rank > 2 responses)

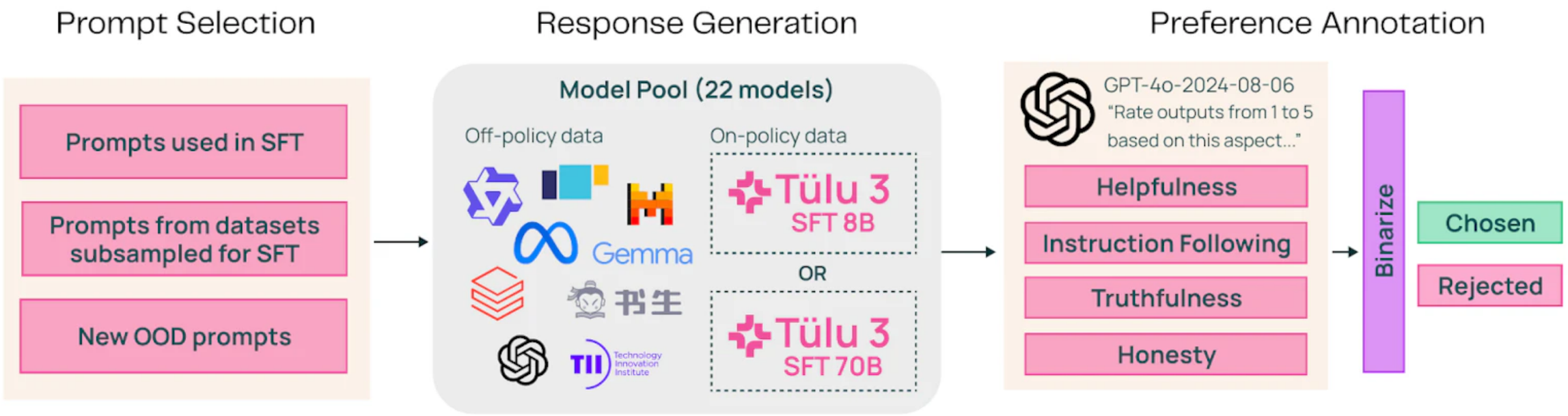
- Approach 1: use human labelers
  - E.g. Open AI, Meta, hire them




- Approach 2: use a strong language model
  - E.g. AlpacaFarm [Dubois et al 2024]: used a GPT model to rate responses generated by Llama



# Large-scale example (AI2 Tulu 3)




Sample four responses from different models for each prompt

# Training the reward model

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

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

Where does this come from?

# Reward model objective

- **Bradley-Terry model** (1952): A probability model for the outcome of pairwise comparisons
- Given items $i, j$, it estimates the probability that the pairwise comparison $i > j$ is true as,

$$\Pr(i > j) = \frac{p_i}{p_i + p_j}$$

# Reward model objective

- Define $p_i = \exp\left(r_\theta(y_i)\right)$:

$$p(y_i \succ y_j) = \frac{\exp(r_\theta(y_i))}{\exp(r_\theta(y_i)) + \exp(r_\theta(y_j))}$$

$$= \frac{1}{1 + (\exp(r_\theta(y_j))/\exp(r_\theta(y_i)))}$$

Divide by $\exp(r_\theta(y_i))$

$$= \frac{1}{1 + \exp\left(-[r_\theta(y_i) - r_\theta(y_j)]\right)}$$

$$= \sigma\left(r_\theta(y_i) - r_\theta(y_j)\right)$$

Sigmoid function $\sigma(z) = \frac{1}{1 + \exp(-z)}$

# Reward model objective

- Likelihood of observing all preferences in the dataset:

$$\mathscr{L}(\theta) = \prod_{(y_i, y_j) \in \mathscr{D}} \sigma(r_\theta(y_i) - r_\theta(y_j))$$

- Maximize likelihood (minimize negative log-likelihood) via the loss:

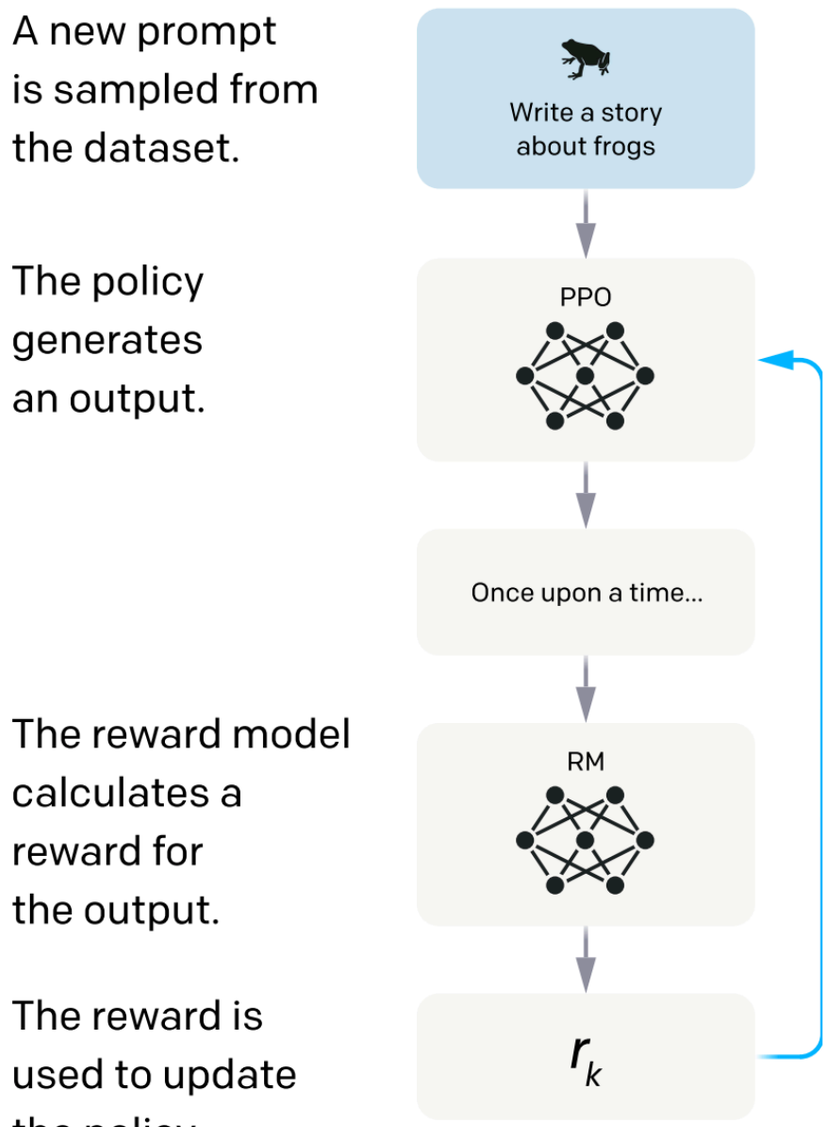
$$\begin{aligned} \mathscr{L}_{NLL}(\theta) &= -\log \mathscr{L}(\theta) \\ &= -\sum_{(y_i, y_j) \in \mathscr{D}} \log \sigma \left( r_\theta(y_i) - r_\theta(y_j) \right) \end{aligned}$$

# Step 3: Reinforcement learning

3. **Reinforcement learning (RL)**: Further fine-tune the language model from step 1 (call it $p_0$) using a reinforcement learning algorithm:

$$p_\theta = RL(p_0, r_\phi, \{x^{(n)}\}_{n=1}^{N''})$$

- Important in practice (*next slides*):
  - Prevent the model $p_\theta$ from moving too far from the original model $p_0$
    - "Catastrophic forgetting", reward hacking



# KL Divergence constraint

- Maximize expected reward subject to a KL divergence penalty:

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

  - Higher $\beta$: more pressure to stay close to the original model
  - Lower $\beta$: more freedom to maximize reward

- Common approach: introduce through a modified reward:

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

Requires keeping around a copy of the original model $p_0$ (“reference policy”)!

# KL Divergence constraint

This reward approximates the KL divergence:

$$D_{\text{KL}}\left(p_\theta(y\,|\,x)\|p_0(y\,|\,x)\right) = \sum_y p_\theta(y\,|\,x)\log\frac{p_\theta(y\,|\,x)}{p_0(y\,|\,x)}$$

$$= \mathbb{E}_{y\sim p_\theta}\log\frac{p_\theta(y\,|\,x)}{p_0(y\,|\,x)}$$

$$\approx \log\frac{p_\theta(\hat{y}\,|\,x)}{p_0(\hat{y}\,|\,x)}$$

where $\hat{y}\sim p_\theta(\,\cdot\,|\,x)$, i.e. a single-sample Monte-Carlo approximation.

# KL Divergence constraint

In summary, we add a reward penalty so that we optimize:

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

The policy that maximizes this objective is:

$$p_\theta(y\,|\,x) = \frac{1}{Z(x)} p_0(y\,|\,x)\exp\left(\frac{1}{\beta} r(x,y)\right)$$

See Korbak et al 2022 or Rafailov et al 2023 for the derivation

# Algorithm variants

**Policy gradient (REINFORCE)**

$$L_{PG} = W_t \log p_\theta(y_t \mid y_{<t}, x)$$

=> gradient = $W_t \nabla_\theta \log p_\theta(y_t \mid y_{<t}, x)$

**Proximal policy optimization (PPO)**

$$L_{PPO} = \min\left(\frac{p_\theta(y_t \mid y_{<t}, x)}{p_{\theta_{old}}(y_t \mid y_{<t}, x)} W_t,\ \text{clip}\left(\frac{p_\theta(y_t \mid y_{<t}, x)}{p_{\theta_{old}}(y_t \mid y_{<t}, x)}(x, y), 1-\epsilon, 1+\epsilon\right) W_t\right)$$

Next: different choices for weights $W_t$

# Weighting variants

Different ways to weight each gradient update based on the rewards:

- Option 1: use discounted cumulative rewards
  - $W_t : R_t = \sum_{k=t+1} \gamma^{k-t} r_k$
- Option 2: use a baseline:
  - $W_t : (R_t - b(s_t))$
  - Common baseline: learn a *state-value function*: $V(s_t)$

# Weighting variants

*State-value function*: predicts how good the state is

$$V(s_t) = \mathbb{E}[R_t]$$

In practice, add a scalar output head to the transformer, collect states and rewards, and minimize an L2 loss:

$$L_{VF} = \|v_\phi(s_t) - R_t\|$$

When used as a baseline, the observed return needs to be better than average in order to result in a positive gradient update:

$$W_t = (R_t - v_\phi(s_t))$$

# General view: advantages

We can define the *advantage* of state and action:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

$$V(s_t) = \mathbb{E}[R_t]$$
$$Q(s_t, a_t) = \mathbb{E}[R_t \mid a_t]$$

Advantage is positive when the action $a_t$ is better than average

We can set $W_t = A(s_t, a_t)$ if we have a way to estimate the advantage

# Estimating advantages

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

Several estimates do not add additional bias when used in place of the true advantage [Schulman et al 2016]:

- $R_t$ (Option 1, discounted return)
- $R_t - b(s_t)$ (Option 2, with baseline)
- $r_t + \gamma V(s_{t+1}) - V(s_t)$ (*Temporal-Difference (TD) Error)*

# Estimating advantages

*Generalized advantage estimation (GAE)* [Schulman 2016] generalizes TD error, allowing for trading off bias and variance:

- $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$ (*Temporal-Difference (TD) Error)*
- $\mathrm{GAE}_t(\gamma, \lambda) = \sum_{t'=t}^{T} (\gamma\lambda)^{\ell} \delta_{t'}$
  - $\mathrm{GAE}_t(\gamma,0) = \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
  - $\mathrm{GAE}_t(\gamma,1) = R_t - V(s_t)$

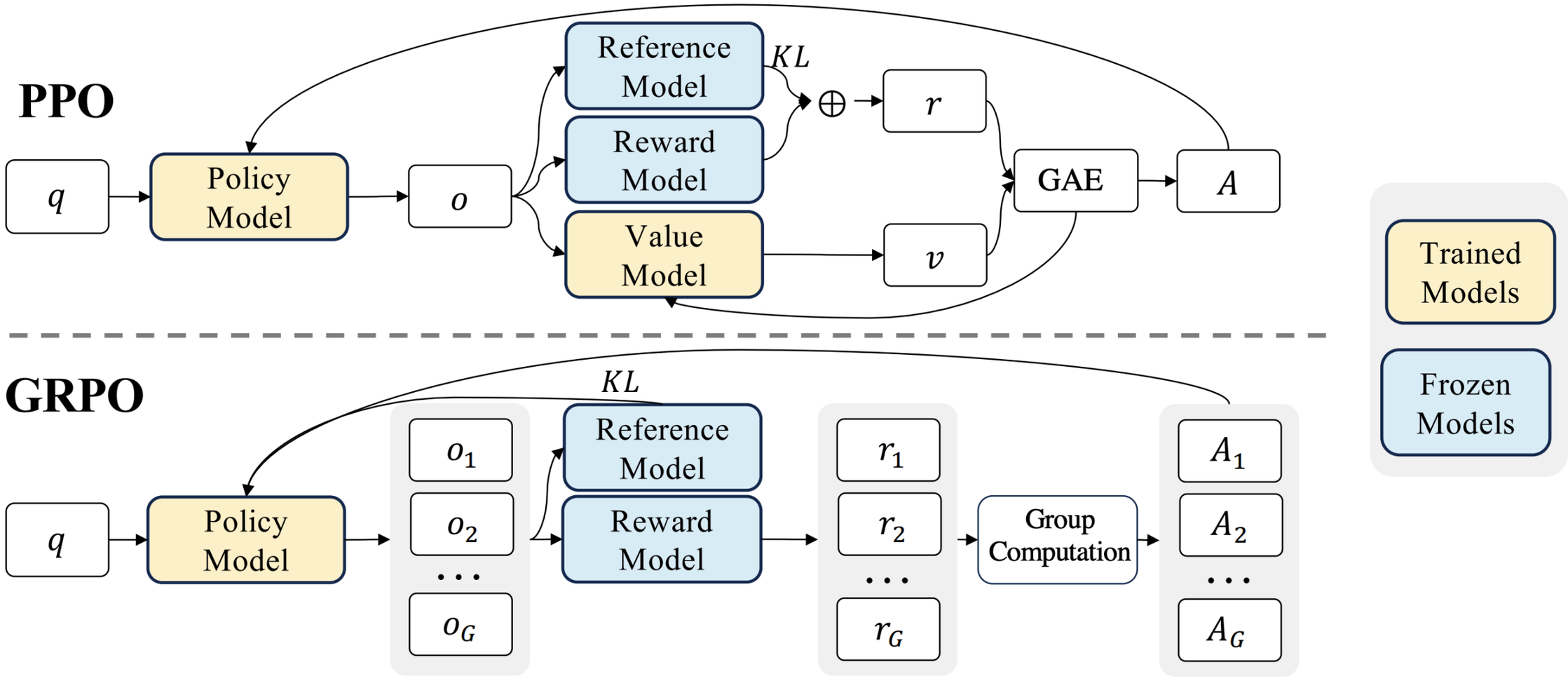
# Weighting variants

Finally, *group-relative policy optimization* (GRPO) computes weights based on drawing multiple samples $y^{(1)}, \ldots, y^{(K)}$

- $$W_t = \sum_{t'=t}^{T} \frac{r_{t'}^{(k)} - \text{mean}}{\text{std}},$$

where mean and standard deviation are computed over the rewards from the multiple samples

# Weighting variants




DeepSeek-Math

# Weighting variants: recap

There are several ways to weight the gradient updates based on the reward. Some require learning a value function $v_\phi(s_t)$

| | Hyperparam | Key idea | Extra model? |
|---|---|---|---|
| **Discounted Return** | Discount factor | Less reward on previous steps | No |
| **GRPO** | Discount factor<br>Number of samples | Draw multiple samples, take the mean / std | No |
| **Value function baseline** | Discount factor | Baseline based on how good a state is | Value function |
| **Generalized advantage estimation** | Discount factor<br>Lambda parameter | Use the value function in more clever ways | Value function |

# Putting it all together

**Given**: initial language model $p_0$, dataset of prompts $\{x^{(n)}\}_{n=1}^{N}$, reward $r(x, y)$

Run a RL algorithm

- Generate a response, $\hat{y} \sim p_\theta(y \mid x)$
- Compute advantages
    - Run reward model, compute KL penalty, possibly run value function
- Compute loss (e.g., policy gradient, PPO), update $p_\theta$ with gradient descent
    - If applicable, also update value function $v_\phi$ with gradient descent

# Note: not specific to ‘human feedback’!

**Given**: initial language model $p_0$, dataset of prompts $\{x^{(n)}\}_{n=1}^{N}$, reward $r(x, y)$

Run a RL algorithm

- Generate a response, $\hat{y} \sim p_\theta(y \mid x)$
- Compute advantages
  - Run reward model, compute KL penalty, possibly run value model
- Compute loss (e.g., policy gradient, PPO) and update $p_\theta$with gradient descent

Task-specific aspect: prompts and reward

Example: x is a math problem
Reward is correct/incorrect answer

# Example: trl

- https://github.com/huggingface/trl

**SFTTrainer**

Here is a basic example of how to use the `SFTTrainer`:

```
from trl import SFTTrainer
from datasets import load_dataset

dataset = load_dataset("trl-lib/Capybara", split="train")

trainer = SFTTrainer(
    model="Qwen/Qwen2.5-0.5B",
    train_dataset=dataset,
)
trainer.train()
```

# Example: trl

- https://github.com/huggingface/trl

**RewardTrainer**

Here is a basic example of how to use the `RewardTrainer`:

```python
from trl import RewardConfig, RewardTrainer
from datasets import load_dataset
from transformers import AutoModelForSequenceClassification, AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
model = AutoModelForSequenceClassification.from_pretrained(
    "Qwen/Qwen2.5-0.5B-Instruct", num_labels=1
)
model.config.pad_token_id = tokenizer.pad_token_id

dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")

training_args = RewardConfig(output_dir="Qwen2.5-0.5B-Reward", per_device_train_batch_size=2)
trainer = RewardTrainer(
    args=training_args,
    model=model,
    processing_class=tokenizer,
    train_dataset=dataset,
)
trainer.train()
```

# Example: trl

- https://github.com/huggingface/trl

### `GRPOTrainer`

`GRPOTrainer` implements the Group Relative Policy Optimization (GRPO) algorithm that is more memory-efficient than PPO and was used to train Deepseek AI's R1.

```python
from datasets import load_dataset
from trl import GRPOTrainer

dataset = load_dataset("trl-lib/tldr", split="train")

# Dummy reward function: count the number of unique characters in the completions
def reward_num_unique_chars(completions, **kwargs):
    return [len(set(c)) for c in completions]

trainer = GRPOTrainer(
    model="Qwen/Qwen2-0.5B-Instruct",
    reward_funcs=reward_num_unique_chars,
    train_dataset=dataset,
)
trainer.train()
```

# Example: verl

- https://github.com/volcengine/verl

**Running a PPO example step-by-step:**

- Data and Reward Preparation
  - Prepare Data for Post-Training
  - Implement Reward Function for Dataset
- Understanding the PPO Example
  - PPO Example Architecture
  - Config Explanation
  - Run GSM8K Example

**Algorithm**

```yaml
algorithm:
  gamma: 1.0
  lam: 1.0
  adv_estimator: gae
  kl_penalty: kl  # how to estimate kl divergence
  kl_ctrl:
    type: fixed
    kl_coef: 0.005
```

- `gemma`: discount factor
- `lam`: Trade-off between bias and variance in the GAE estimator
- `adv_estimator`: Support `gae`, `grpo`, `reinforce_plus_plus`, `rloo`
- `kl_penalty`: Support `kl`, `abs`, `mse` and `full`. How to calculate the kl divergence between actor and reference policy. For specific options, refer to core_algos.py .

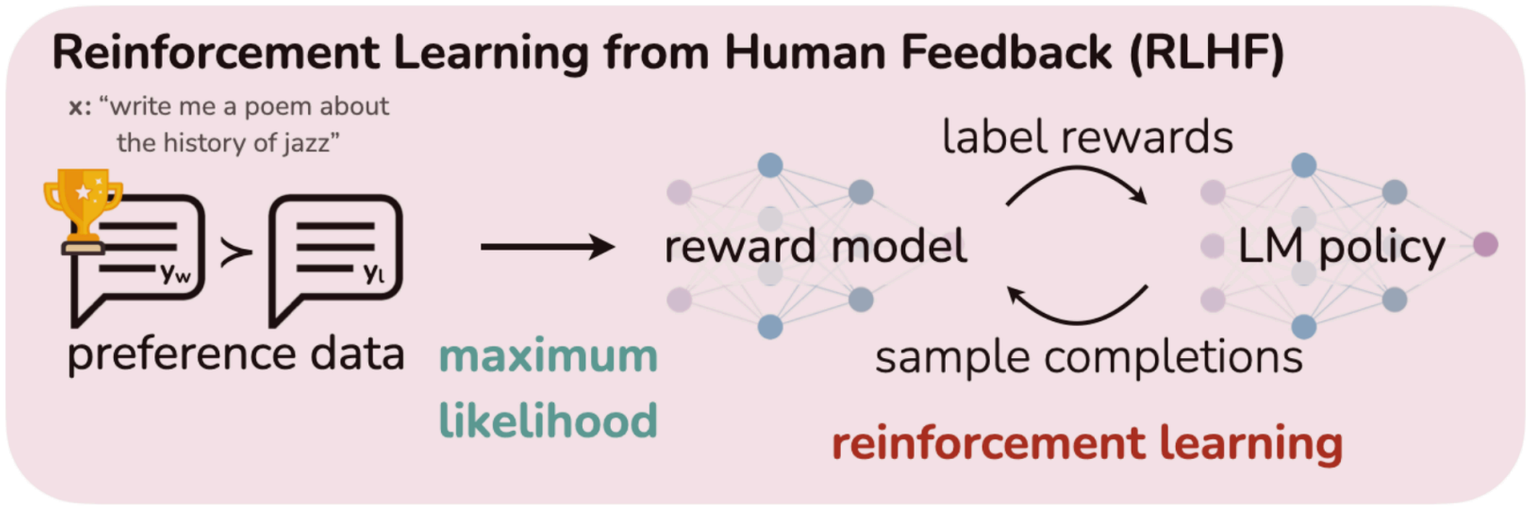
# Today's lecture

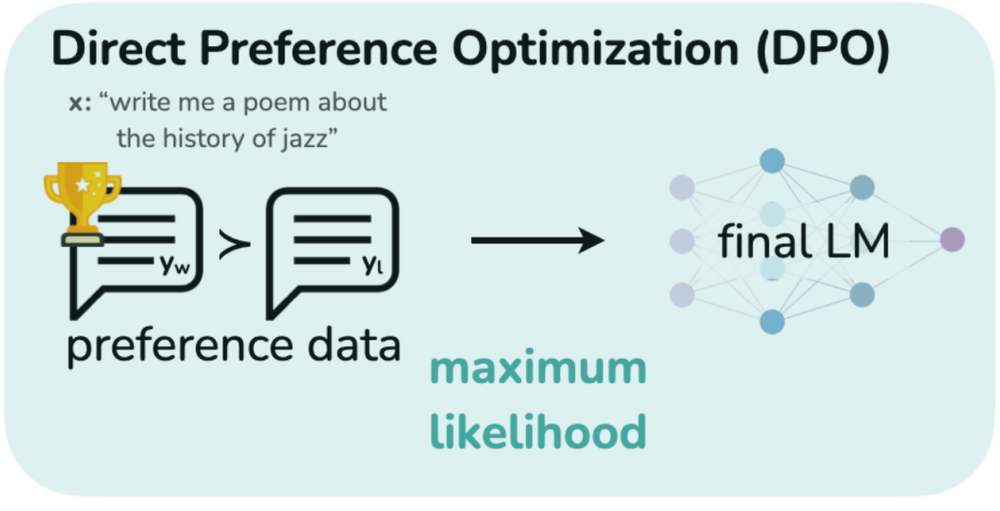
- Language model reinforcement learning pipeline
- **Direct preference optimization**

# Direct Preference Optimization

[Rafailov et al 2023]

- For the specific case of a preference reward, can we skip having to train a separate reward model?



# Direct Preference Optimization

[Rafailov et al 2023]

- Recall that under the Bradley Terry model we have:

$$p(y_i \succ y_j) = \frac{\exp(r(y_i))}{\exp(r(y_i)) + \exp(r(y_j))}$$

$$= \sigma\left(r(y_i) - r(y_j)\right)$$

- And that the optimal KL-constrained policy is:

$$p_\theta(y \mid x) = \frac{1}{Z(x)} p_0(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

# Direct Preference Optimization

[Rafailov et al 2023]

- We can rewrite the optimal KL-constrained policy in terms of its reward:

$$p_*(y \mid x) = \frac{1}{Z(x)} p_0(y \mid x) \exp\left(\frac{1}{\beta} r(x, y)\right)$$

$$r_*(x, y) = \beta \log \frac{p_*(y \mid x)}{p_0(y \mid x)} + \beta \log Z(x)$$

- Plug this into the Bradley-Terry model:

$$p_*(y_1 \succ y_2) = \frac{1}{1 + \exp\left(\beta \log \frac{p_*(y_2 \mid x)}{p_0(y_2 \mid x)} - \beta \log \frac{p_*(y_1 \mid x)}{p_0(y_1 \mid x)}\right)}$$

- Now replace $p_*$ with $p_\theta$ and use the maximum likelihood objective for the Bradley-Terry model to learn the parameters $\theta$

# Direct Preference Optimization

[Rafailov et al 2023]

$$\mathscr{L}_{\text{DPO}}(p_\theta; p_0) =$$

$$-\mathbb{E}_{(x,y_w,y_l)\sim\mathscr{D}}\left[\log\sigma\left(\beta\log\frac{p_\theta(y_w\,|\,x)}{p_0(y_w\,|\,x)} - \beta\log\frac{p_\theta(y_l\,|\,x)}{p_0(y_l\,|\,x)}\right)\right]$$

# Direct Preference Optimization

[Rafailov et al 2023]

$$\nabla_\theta \mathscr{L}_{\text{DPO}}(p_\theta; p_0) =$$

$$-\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathscr{D}} \left[ \sigma\left(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)\right) \left[ \nabla_\theta \log p(y_w \,|\, x) - \nabla_\theta \log p(y_l \,|\, x) \right] \right]$$

where $\hat{r}_\theta(x, y) = \beta \log \dfrac{p_\theta(y \,|\, x)}{p_0(y \,|\, x)}$ is the “implicit reward”.

# Comparison

| | Data | Reward | Data | Models |
|---|---|---|---|---|
| **DPO** | Offline | Preference | Preference pairs | Policy<br>Reference |
| **RLHF with PPO** | Online | Preference | Preference pairs for RM<br><br>Prompts for RL | Policy<br>Reference<br>Value function<br>Reward |
| **General Online RL** | Online | Any | Prompts for RL<br><br>+ Depends on reward | Policy<br>+ depends on objective, advantages, reward |

# Example: trl

- https://github.com/huggingface/trl

**DPOTrainer**

`DPOTrainer` implements the popular Direct Preference Optimization (DPO) algorithm that was used to post-train Llama 3 and many other models. Here is a basic example of how to use the `DPOTrainer`:

```python
from datasets import load_dataset
from transformers import AutoModelForCausalLM, AutoTokenizer
from trl import DPOConfig, DPOTrainer

model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2.5-0.5B-Instruct")
dataset = load_dataset("trl-lib/ultrafeedback_binarized", split="train")
training_args = DPOConfig(output_dir="Qwen2.5-0.5B-DPO")
trainer = DPOTrainer(
    model=model,
    args=training_args,
    train_dataset=dataset,
    processing_class=tokenizer
)
trainer.train()
```

# Today's lecture

- **Language model reinforcement learning pipeline**
  - Example: RLHF
    - Supervised fine-tuning
    - Reward modeling
    - Reinforcement learning
- **Direct preference optimization**

# Questions?