TEXAS
The University of Texas at Austin

RLHF FOUNDATIONS

2026F INF385T: Post-Training of LLMs | Week 8

CHAN YOUNG PARK
Assistant Professor, iSchool, School of Computing, The University of Texas at Austin

Announcements

- HW1 is due this Thursday (Oct 15) 11:59pm
- HW2 (preference data + DPO) will be released next week
- Project checkpoint in Week 10 (5-minute verbal update per team)

Recap

Week 6: which parameters receive updates? → parameters

Week 7: which behaviors receive demonstrations? → examples

Week 8: what if we can only compare? → preferences

Last week ended with a question

How can “A is better than B” become a training signal?

1. Comparison → reward

Fit a scalar reward model r(x, y) so that preferred responses score higher

Bradley–Terry reward model



2. Reward → policy

Change the LM so that its own samples score higher, without drifting too far

PPO with a KL clipping

Week 8 overview

- **Why preferences**
  InstructGPT: how do comparisons fit into the pipeline? (Ouyang et al., 2022)
- **Reward models**
  Bradley–Terry: how does a comparison become a scalar? (Bradley & Terry, 1952)
- **Policy optimization**
  PPO: how do we optimize a policy against a learned reward? (Schulman et al., 2017)

What do demonstrations miss?

WHY GO BEYOND SFT?

Three limits of demonstrations

Writing << judging

Writing an ideal answer is slow and needs expertise

Picking the better of two answers is fast

+ possible even when no one can write the ideal one

Positive-only signal

SFT says “do this”
never “not that”

No gradient pushes down the bad responses the model already produces

Gold prefixes only

Teacher forcing trains on the dataset’s prefixes

At inference, the model conditions on its own (often wrong) tokens

Comparisons are cheaper to collect and can be made on the model’s own outputs

InstructGPT (Ouyang et al., 2022)

Training language models to follow instructions with human feedback

Long Ouyang* Jeff Wu* Xu Jiang* Diogo Almeida* Carroll L. Wainwright*

Pamela Mishkin* Chong Zhang Sandhini Agarwal Katarina Slama Alex Ray

John Schulman Jacob Hilton Fraser Kelton Luke Miller Maddie Simens

Amanda Askell† Peter Welinder Paul Christiano*†

Jan Leike* Ryan Lowe*

OpenAI

InstructGPT in three steps

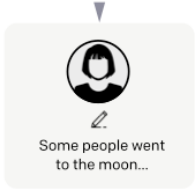
Step 1

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.

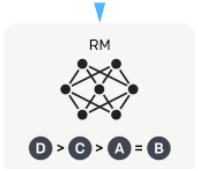
A prompt and several model outputs are sampled.



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.

Ouyang et al. (2022). Training language models to follow instructions with human feedback

InstructGPT in three steps

Step 1. SFT

Labelers write demonstrations

Fine-tune GPT-3 → supervised policy
~13K prompts

Step 2. Reward model

Sample K outputs per prompt

Labelers rank them

Train RM on all pairs
~33K prompts

Step 3. PPO

Policy samples a response

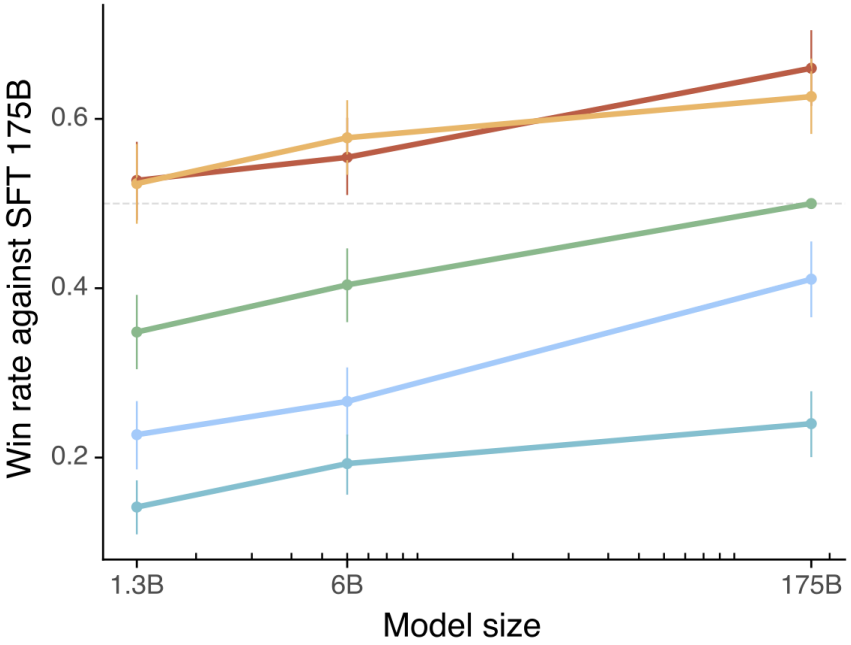
RM scores it

PPO updates the policy
~31K prompts

did it work?

1.3B > 175B

1.3B InstructGPT outputs preferred over 175B GPT-3 outputs



- Fewer hallucinated facts in closed-domain tasks (small toxicity gains)
- Regressions on some public NLP benchmarks (“alignment tax”)

What a preference row looks like

```
{
  "prompt":    "Rewrite this in exactly
                two sentences: ...",
  "chosen":    "Here is a two-sentence
                rewrite: ...",
  "rejected":  "Sure! I'm happy to help!
                Here are three ...",
  "strength":  "significantly better"
}
```

- Pairwise: one chosen, one rejected
- K-way ranking: InstructGPT ranks K = 4-9 outputs → C(K, 2) pairs
- (optional) Graded strength: Llama 2 asks how much better
- (optional) Multi-aspect: separate helpfulness and safety labels

Human agreement is context for RM accuracy

~73%

InstructGPT training labelers agreeing with one another

- If labelers disagree on ~1 in 4 pairs, what does 90% RM accuracy measure?
- Accuracy depends on the reference (a single rater, majority vote, or an adjudicated label)
- Disagreement = noise, specification gap, plural values (Week 4)

How does a comparison become a scalar?

BRADLEY–TERRY REWARD MODELS

A running example

Prompt: “Explain photosynthesis to a 10-year-old.”

Response A

“Plants make their own food. They use sunlight, water, and air…”

Response B

“In the membranes of the chloroplast, light-dependent reactions…”

10 labelers: **7 pick A, 3 pick B**

Goal: give each response a score that explains these votes

The idea: every response has a hidden score

- Think of sports teams with a strength rating
- Big rating gap → the stronger team usually wins
- Small rating gap → close to a coin flip
- Upsets still happen

P(A beats B)
depends (only) on
r(A) − r(B)

A reward model learns
these hidden scores r(x) from comparisons

The Bradley–Terry model

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

chosen (“win”) response

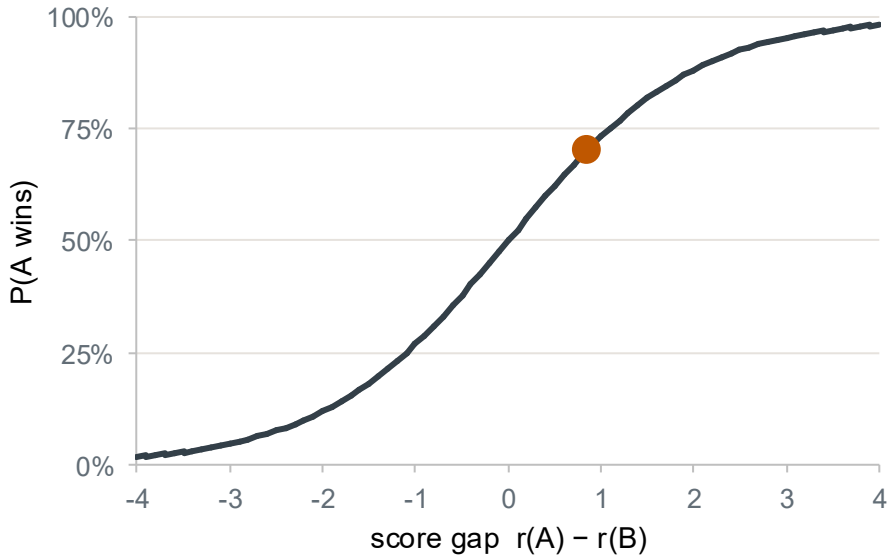
rejected (“lose”) response

hidden score of response y for prompt x

Only the gap between two scores enters the model

Sigmoid
= squashes any number into a probability between 0 and 1

From score gap to win probability



P(A wins)	Score gap
50%	0
70%	**0.85**
88%	2
95%	3

Our example: A wins 7 of 10

→ **r(A) − r(B) ≈ 0.85**

gap = log(p/(1-p))

Where does the sigmoid come from?

Impression of A
= r(A) + random mood

Impression of B
= r(B) + random mood



The labeler picks the higher impression

Even if r(A) > r(B), a bad mood sometimes flips the choice

With one specific noise shape (Gumbel),
P(pick A) = σ(r(A) − r(B)) exactly

Three hidden assumptions

Quality is one number

= Helpful and safe collapse into one score

No rock-paper-scissors
(A > B and B > C, then A > C)

Each judgment is independent

= Every choice is a fresh coin flip

Real labelers get tired, anchor on what they saw first

Everyone shares one scorecard

= Labelers differ only by random mood

Real people have genuinely different tastes

A big gap means “consistent” != “better”

Pair	Votes for A	Score gap
A is slightly better and everyone notices	9 of 10	2.2
A is much better for most, but some prefer B’s style	6 of 10	0.4

The noticeable “slightly better” pair gets the bigger reward gap

Score gap = how consistently people choose and not how much better it is

Only differences matter

Similar to height, the gap is the same measured from the floor or from sea level

r(A)	r(B)	Gap	P(A wins)
0.85	0	0.85	70%
100.85	100	0.85	70%
−5	−5.85	0.85	70%

However, the data never says where zero is for a prompt

Consequences of a free zero point

Don’t compare across prompts

Photosynthesis answers: 5, 6
Poem answers: −3, −2
≠ the photosynthesis answers are better

Normalize before RL

Scores can drift anywhere

InstructGPT shifted the RM so labeler demos average 0

A reward model learns a ranking within a prompt (not an absolute score)

How wrong is the model on this pair? (loss)

The human picked A. How much probability did the model give A?

Model's P(A wins)	Loss −log P
90%	0.11
70%	0.36
50%	0.69
10%	2.30

Averaged over all pairs:

$$\text{Loss}(\varphi) = -\log \sigma(r_\varphi(x, y_w) - r_\varphi(x, y_l))$$

= logistic regression on the score gap

Small penalty when the model is aligned with the human's choice

How much does each pair move the model? (gradient)

gradient = 1 − (model's probability on the human's choice)

Model's P on the human's choice	Gradient	Meaning
95%	0.05	already right → barely move
50%	0.50	Unsure → move a medium amount
5%	0.95	confidently wrong → move the most

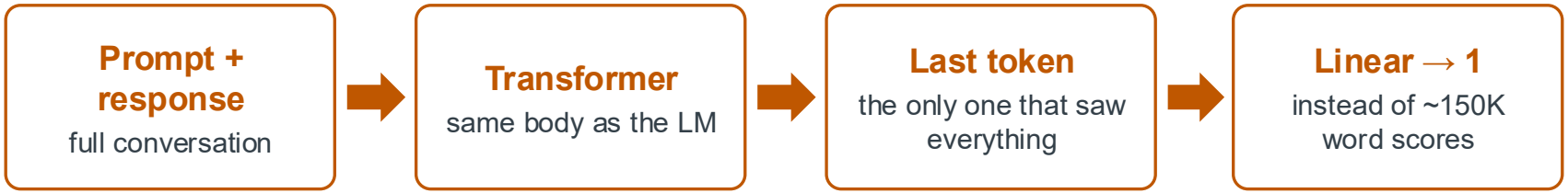
- Direction: raise the chosen score, lower the rejected score
- The penalty is 0 only at an infinite gap → long training memorizes → usually train for 1 epoch
- Label noise gets the most update

Worked example

1. **Scores:** r(chosen) = 1.2, r(rejected) = 0.4
2. **Gap:** 1.2 − 0.4 = 0.8
3. **Probability:** σ(0.8) ≈ 0.69
4. **Penalty:** −log 0.69 ≈ 0.37
5. **Push:** 1 − 0.69 = 0.31

	Right	Wrong
Gap	+0.8	−0.8
Loss	0.37	1.17
Gradient	0.31	0.69

Architecture



→ **r(x, y): one number**

- **start from SFT.** It already understands the kind of responses it will grade
- **Could be smaller than the policy.** RL calls it constantly. InstructGPT: 6B RM for a 175B policy

Reward model training in code

```
rm = AutoModelForSequenceClassification.from_pretrained(
        "Qwen/Qwen2.5-0.5B-Instruct", num_labels=1)
rm.config.pad_token_id = tok.pad_token_id

r_w = rm(**chosen_batch).logits.squeeze(-1)    # [B]
r_l = rm(**rejected_batch).logits.squeeze(-1)  # [B]
loss = -F.logsigmoid(r_w - r_l).mean()
acc  = (r_w > r_l).float().mean()
# TRL's RewardTrainer wraps the same loss
```

RM accuracy is evidence for which claim?

Held-out pair accuracy

Claim: ranks pairs like the labelers did

Best-of-N quality

Claim: picks good samples from my policy

Requires N× inference

Post-RL policy quality

Claim: is a useful optimization target

Requires a full RL run

Probing for spurious features

Probe	Pair differs only in	A good RM should…
Length	same answer + filler sentences	barely change
Format	prose vs. bullet list	barely change
Sycophancy	adds flattery / agrees with a false premise	score lower
Correctness	one factual or arithmetic error	score clearly lower

What exactly are we optimizing?

FROM REWARD TO POLICY

An LM is a policy



Policy = LM (a probability for every next token)

Episode = one full response

A toy example: three possible responses

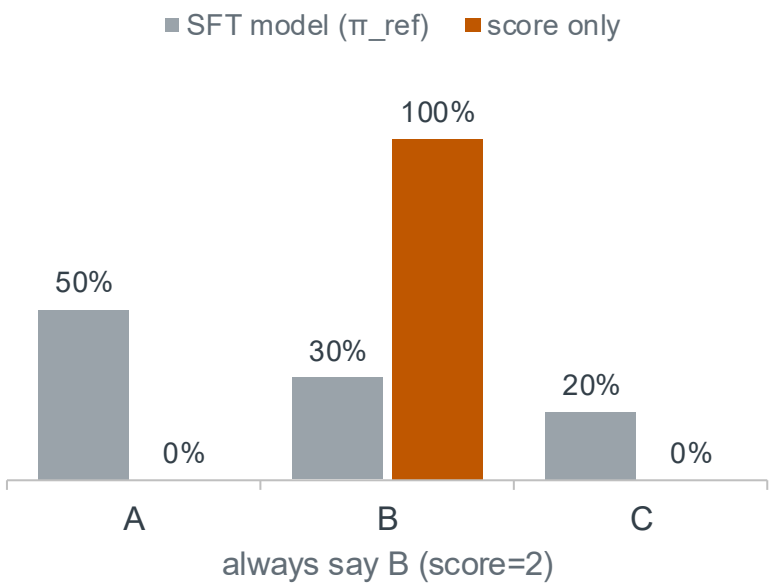
Response	SFT model says it	RM score
A (okay answer)	50%	1
B (great answer)	30%	2
C (weak answer)	20%	0



Average score of π: **$0.5\times1 + 0.3\times2 + 0.2\times0 = 1.1$**

Goal: update the model so its average score goes up

Why not just maximize the score?



The RM is only reliable nearby
When far from its training data it can be fooled

Diversity collapses
Every answer becomes the same single style

Skills can get lost
Fluency and knowledge live in the SFT model

Goal: We want a higher score without drifting too far from the SFT model

The KL-regularized objective

$$\max \quad E[\, r(x, y) \,] \quad - \quad \beta \cdot KL(\, \pi_\theta \,\|\, \pi_{ref} \,)$$

Raise the score

average RM score of what the model writes

Don't drift too far

how surprised the SFT model is by what the new model writes

β is a budget for how far you trust the reward model

small β — trust the RM (move far)

large β — distrust the RM (stay close)

In practice, every token pays a small fee

	Plants	make	food	RM: +2.0
$\beta \cdot KL(\pi_\theta \| \pi_{ref})$	−0.02	+0.01	−0.05	

Total reward = 2.0 − 0.06 = **1.94**

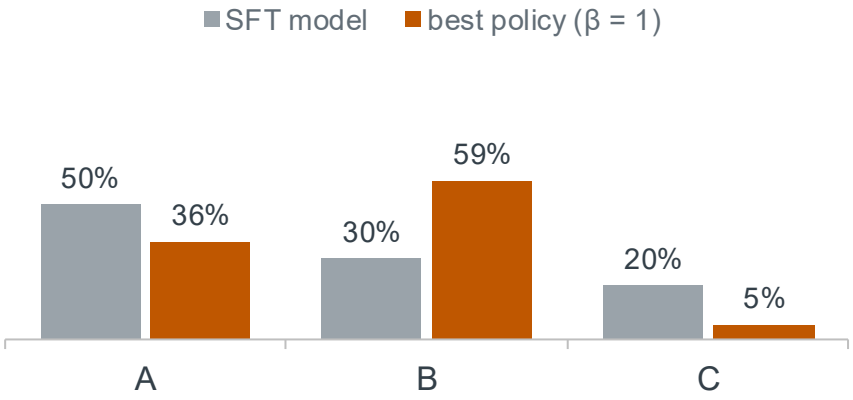
fee per token = β × log(new model's probability ÷ SFT model's probability)

The best policy = reweighted SFT model

The best policy has a closed form:

$$\pi^*(y \mid x) = (1 / Z(x)) \cdot \pi_{ref}(y \mid x) \cdot \exp(r(x, y) / \beta)$$

$Z(x)$ = sum of $(\pi_{ref} \cdot \exp(r/\beta))$ over every possible response



- Keep what the SFT model already says. boost what scores high
- Average score 1.1 → 1.53

Catch: in practice, we can't compute $Z(x)$

Simplest optimizer: Best-of-N

Sample N from the SFT model → **Score** each with the RM → **Keep** the best one



- Toy world, N = 4
  - B: $1 - 0.7^4 \approx 76\%$
  - A: $0.7^4 - 0.2^4 \approx 24\%$
  - C: $0.2^4 \approx 0.2\%$
- average 1.1 → 1.76
- No training needed, but N× the generation cost
- Llama 2 rejection sampling: fine-tune on Best-of-N outputs

How do we update the policy?

POLICY GRADIENTS → PPO

REINFORCE = try, score, update

J = avg score of model responses = reward * generation probability

We want to increase J → gradient ascent + update policy

$$\nabla_\theta J = E_{y\sim\pi\theta} [R(x, y) \cdot \nabla_\theta \log \pi_\theta(y \mid x)]$$

1. Try
sample a response from the current model

2. Score
get R(x, y) from the reward

3. Update
make high-scoring responses more likely

Sampled	Score R	Update
B: great answer	2	raise B a lot
A: okay answer	1	raise A a little
C: weak answer	0	no change

SFT is REINFORCE with R = 1

SFT

$1 \cdot \nabla \Sigma_t \log \pi_\theta(y^*_t \mid x, y^*_{<t})$

y*: written by the dataset
weight: always 1

REINFORCE

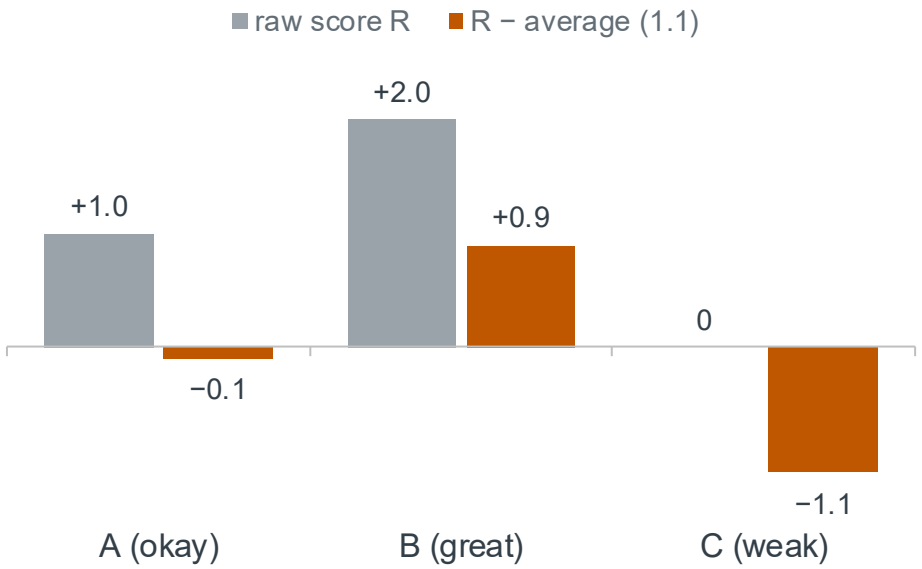
$R \cdot \nabla \Sigma_t \log \pi_\theta(y_t \mid x, y_{<t})$

y: written by the model itself
weight: reward R

Same formula/objective

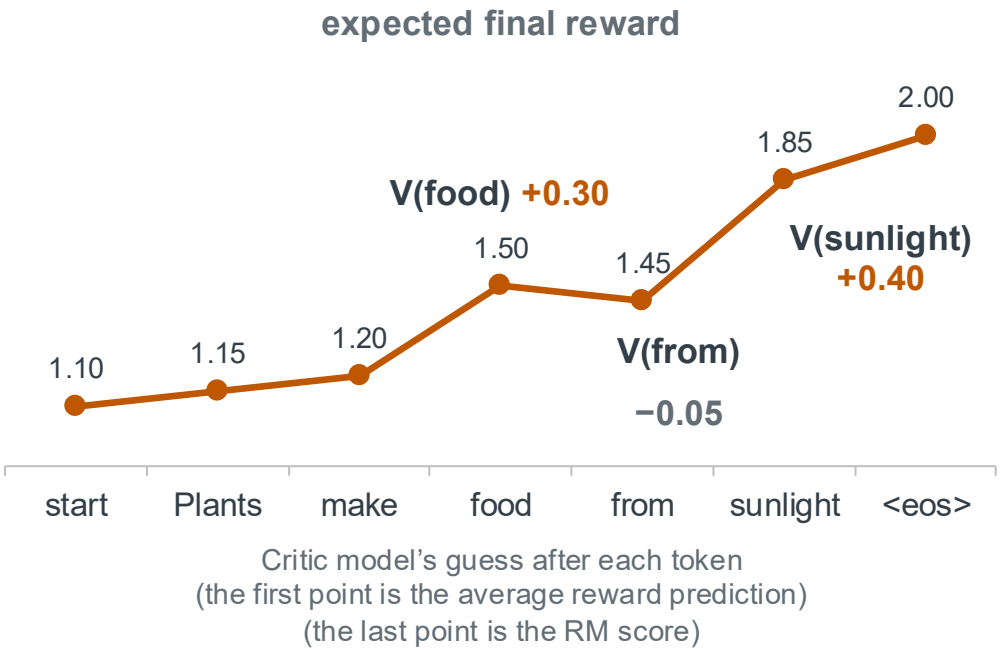
Only two differences are (1) on-policy, (2) using RM

From score to advantage (subtract a baseline)



- **Problem:** all scores ≥ 0, so every sample gets pushed up, even the okay one
- **Fix:** subtract the expected score (1.1)
- Now B goes up, A/C goes down
- How to get the expected score?
  → critic model's prediction

Critic model: predicts per-token reward



Critic model's guess after each token
(the first point is the average reward prediction)
(the last point is the RM score)

Critic = Value model V(s)

"given what's written so far, what final score do I expect?"

Value/Advantage of a token

= how much it moved the expected score

Why PPO needs clipping (aka a leash)

1. Samples are expensive

Generating responses is the slow part

So we reuse one batch for several updates
And provide an adjustment term (ratio)

2. Mistakes snowball

SFT always learns from a fixed dataset

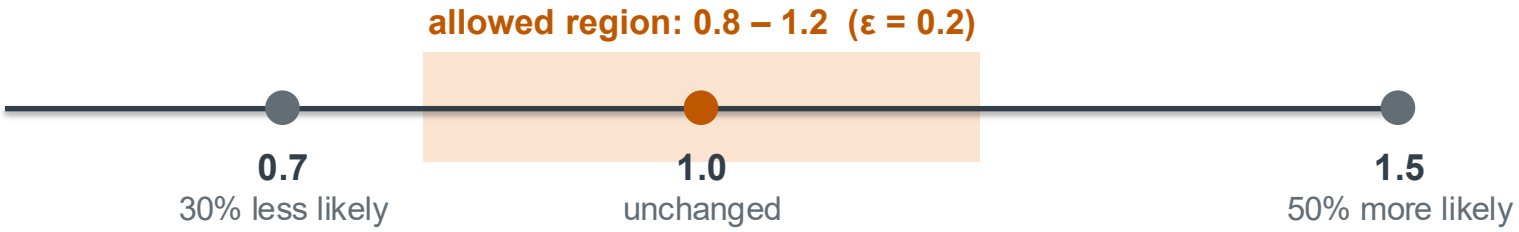
RL learns from the model's own outputs

One bad step → worse outputs → worse training data → …

PPO: clip how far each token can move (Schulman et al., 2017)

$$\rho_t = \pi_\theta(\text{token}) / \pi_{old}(\text{token})$$

how much more (or less) likely this token has become since we sampled



Inside the zone: normal update
Past the edge in the correct direction: this token's gradient becomes 0

The four cases

	Already moved past 20% the right way	Moved the wrong way
B: great answer (A > 0)	ρ = 1.25 → stop raising	ρ = 0.9 → keep raising
C: weak answer (A < 0)	ρ = 0.75 → stop lowering	ρ = 1.1 → keep lowering

Clipping stops overshooting.
But it never stops correcting

The PPO loop



↻ **repeat with the updated policy**

* The policy and the critic are trained together

What to monitor during PPO

Metric	Healthy	Warning sign
Mean reward	rises, then plateaus	rises fast while samples look worse
KL(π_θ ll π_{ref})	grows slowly, bounded	explodes
Response length	stable or modest change	grows every step
Clip fraction	small and stable	large or rising
Entropy	slow decline	collapse → repetitive text

Overhead: four models in memory

Reference π_{ref}
anchor
basis for the KL fee
frozen

Reward r_{φ}
judge
gives the final score
frozen

Policy π_{θ}
player
writes responses
trained

Value V_{ψ}
commentator
predicts the score
trained

Week 9: DPO
drops the judge, the critic, sampling

Week 11: GRPO, RLVR
drops the critic (average over a group instead)
RLVR swaps the judge for a checker

Train and probe a reward model

LAB 8

Lab 8: plan for today

Time	Part
0:00–0:20	Part 1. Be the annotator (10 UltraFeedback pairs)
0:20–0:30	Class results (agreement, our majority vs. GPT-4)
0:30–0:40	Task 1. From votes to Bradley-Terry rewards
0:40–0:50	Task 2. Train a LoRA reward model
0:50–1:05	Task 3. Held-out accuracy and a length baseline
1:05–1:15	Microsoft Form + wrap-up