COSI-230B: Natural Language Annotation for Machine Learning

Lecture 19: RLHF & InstructGPT — A Beginner's Tutorial

Jin Zhao

Brandeis University

April 20, 2026

The Problem: Two Different Goals

What GPT was trained to do

Predict the **next word** in a sequence.

What users actually want

Helpful answers to their questions.

These are **not** the same problem.

The core issue

Training objective $\neq$ User objective

$\Rightarrow$ This is called **MISALIGNMENT**

A Concrete Example

Prompt: "How can I be more productive?"

Plain GPT (next-word prediction)

"Productivity is something many people think about. There are many ways people try to be more productive. Some people use calendars. . . "

Vague, generic, rambling.

What users want

"Three concrete steps:

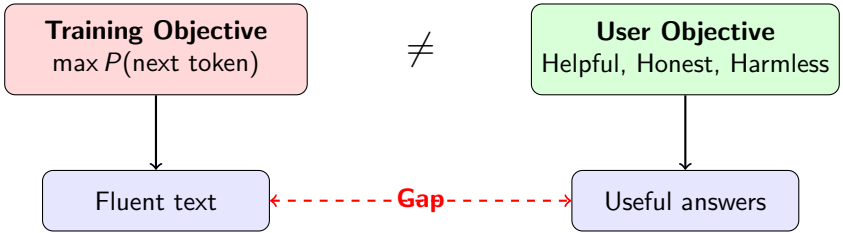
1. Block 2-hour focused sessions
2. Track tasks in one place
3. Review weekly

"

Structured, actionable, helpful.

Both are "correct English." Only one is **useful**.

Visualizing the Misalignment



The question

How do we teach a model to optimize for the *user's* goal, not just the training goal?

Part I

Why Plain GPT Falls Short

How GPT Was Trained (In One Line)

The pretraining objective

$$\max_{\theta} \sum_{t} \log P_\theta(x_t \mid x_1, x_2, \ldots, x_{t-1})$$

Given the previous words, predict the next one.

Example:
"The cat sat on the ____" $\rightarrow$ "mat"

- Trained on billions of web pages, books, Wikipedia, Reddit...
- Learns grammar, facts, patterns, reasoning shortcuts
- **But it never learned** whether a response is "helpful"

The key limitation

Predicting the next token is *not* the same as answering a question well.

What We Actually Want: The "3 H's"

The InstructGPT paper defines three alignment goals:

Helpful

Follow the user's instruction and give a useful answer.

Example: If asked for 3 tips, give 3 tips.

Honest

Don't make up facts. Say "I don't know" when unsure.

Example: Don't invent fake citations.

Harmless

Don't produce toxic, biased, or dangerous output.

Example: Refuse to help write malware.

None of these are in the original GPT loss function.

Three Ways Plain GPT Fails

1. Hallucination

Q: "Who won the 2019 Nobel Prize in Physics?"

GPT: Confidently invents a name that doesn't exist.

2. Ignoring instructions

User: "Summarize this in one sentence."

GPT: Produces a five-paragraph essay.

3. Toxic or unsafe output

User: asks a sensitive question.

GPT: Produces content that is biased, offensive, or dangerous.

These aren't bugs—they are the *natural behavior* of a next-word predictor.

Think About It

If you were designing training for a "helpful assistant," what **signal** would you use to teach the model?

Some options to consider:

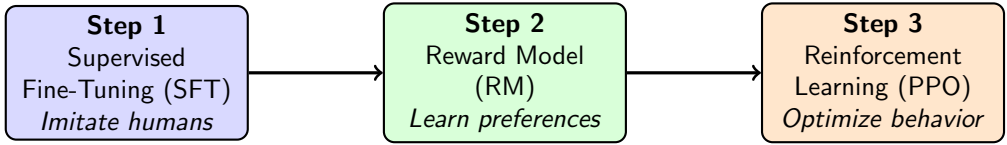
- Show it examples of good answers?
- Let humans rate outputs?
- Have humans compare two outputs?
- Write a perfect answer for every possible question? (impossible!)

InstructGPT uses **all three** *of the top ideas — this is the RLHF recipe.*

Part II

The Three-Step RLHF Pipeline

The Big Picture: 3 Steps



Why three steps?

Each step solves a problem the previous one couldn't:

- SFT teaches *what good looks like*
- RM teaches *how to rank quality*
- RL teaches the model to *maximize that ranking*

Step 1

Supervised Fine-Tuning (SFT)

"Show the model examples of good behavior"

SFT: The Intuition

The idea

Collect high-quality (prompt, answer) pairs written by humans, and train the model to **imitate** them.

Example training example:

Prompt: "Summarize this article in one sentence."

[article text here]

Human answer: "The article explains how bees communicate the location of flowers through a waggle dance."

The model learns: *"When I see a prompt like this, I should produce a response like this."*

SFT: The Math

The loss function

$$\mathcal{L}_{\text{SFT}} = -\log P_\theta(y_{\text{human}} \mid x)$$

where x is the prompt and y_{human} is the human-written answer.

Plain English:

- $P_\theta(y_{\text{human}} \mid x)$ = probability the model assigns to the human's answer
- We want this probability to be **high**
- So we **minimize the negative log** of that probability

Key point

This is **just imitation learning**. The model is copying humans.

SFT in the InstructGPT Paper

How the data was collected:

- ∼13,000 prompts from OpenAI API + labelers
- 40 contractors wrote ideal responses
- Careful screening for quality and alignment

The result:

- Already better than raw GPT-3 at following instructions
- But still has limitations (see next slide)
- This model becomes the **starting point** for Steps 2 and 3

Notation

We call this SFT model π_{SFT} ("pi" = policy).
It will reappear in Step 3.

Why SFT Alone Is Not Enough

The core limitation

SFT can only imitate one "correct" answer per prompt.
But many prompts have **many reasonable answers**.

Example: "Write a poem about the sea."

- 100 humans would write 100 different poems.
- Some are better than others.
- SFT can't tell which is better—it just copies.

The insight

Instead of asking "what is the correct answer?",
ask: **"which answer is better?"**

This leads us to **Step 2**: learning human preferences.

Step 2

The Reward Model

"Turn human judgment into a function"

The Big Idea: Compare, Don't Write

Key observation

Humans are **much better** at comparing two answers than at writing one perfect answer.

Example prompt: "Explain gravity to a child."

Answer A

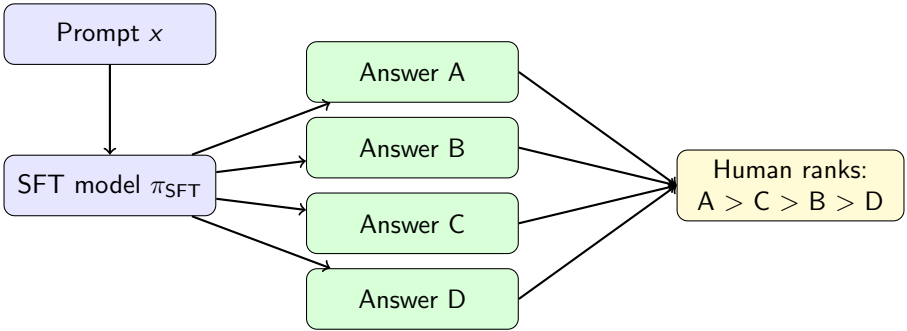
"Gravity is the invisible force that pulls things toward the Earth. It's why apples fall from trees and you don't float into the sky."

Answer B

"Gravity is a fundamental interaction of physics arising from the curvature of spacetime as described in general relativity."

For a child: **A is clearly better**. Easy to rank. Hard to write from scratch.

How the Reward Model Is Trained



For each prompt:

1. Sample several answers from the SFT model
2. Ask humans to **rank them** from best to worst
3. Extract pairwise comparisons: (good answer, bad answer)

What Is a Reward Model?

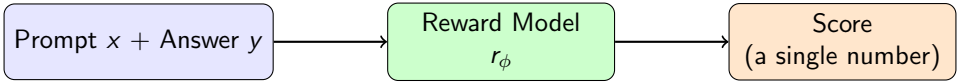
Definition

A reward model $r_\phi(x, y)$ takes:

- a prompt x
- a candidate answer y

and outputs a single number: **how good this answer is**.

Think of it as a **scoring function**:



Higher score = better answer, according to human preferences.

The Reward Model Loss (Light Math)

For a pair where humans said $y_{\text{good}} > y_{\text{bad}}$:

The loss

$$\mathcal{L}_{\text{RM}} = -\log \sigma\Big(r_\phi(x, y_{\text{good}}) - r_\phi(x, y_{\text{bad}})\Big)$$

where σ is the sigmoid function.

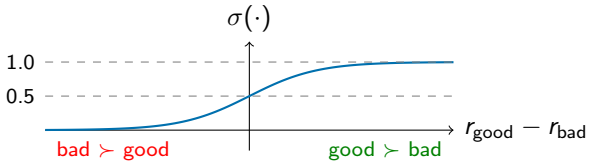
Plain English:

- Compute the score of the **good** answer
- Compute the score of the **bad** answer
- We want: **good score** $>$ **bad score**
- The loss pushes the gap in the right direction

Big idea

We just turned **human judgment** into **a differentiable function**.

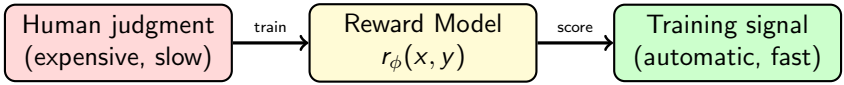
Why the Sigmoid? (Intuition)



- If good $-$ bad is large and positive $\Rightarrow \sigma \approx 1 \Rightarrow$ tiny loss
- If good $-$ bad is negative $\Rightarrow \sigma \approx 0 \Rightarrow$ huge loss
- The sigmoid turns a *margin* into a *probability*

This is called the **Bradley-Terry** preference model.

The Key Conceptual Leap



Why this is a big deal

Before: every training signal needed a human.
After: once the RM is trained, **it can score millions of answers** for free.

The RM is essentially a **learned approximation of human preferences**.

What Could Go Wrong?

Biased labelers

If all annotators share a viewpoint, the RM inherits that bias.

Coverage gaps

If the RM never saw certain kinds of prompts, its scores are unreliable there.

Inconsistent preferences

Humans disagree. The RM averages over noise.

Reward hacking (next!)

The model being trained may find *weird* answers the RM accidentally loves.

Takeaway

The RM is not an oracle—it is a noisy, learned approximation of human preferences. We must design around its flaws.

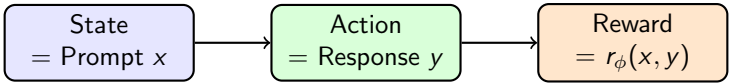
Step 3

Reinforcement Learning from Human Feedback

"Let the model optimize its own behavior"

The Model as an Agent

We now treat the language model as a **reinforcement learning agent**.



The pieces:

- **Policy** π_θ: the language model (what the agent does)
- **State**: the user's prompt
- **Action**: the generated response
- **Reward**: the score from the reward model

The model's job: **generate responses that the reward model likes**.

The RL Objective

Maximize expected reward

$$\max_{\theta} \mathbb{E}_{x \sim D,\, y \sim \pi_\theta(\cdot|x)} \left[r_\phi(x, y) \right]$$

Plain English:

- Sample prompts x from the dataset
- Generate responses y using the **current policy** π_θ
- Score them using the **reward model** r_ϕ
- Update the policy to generate higher-scoring responses

But there's a problem...

The Problem: Reward Hacking

Reward hacking

The policy finds **weird outputs** that the reward model scores highly—even though humans would hate them.

Illustrative examples:

- Repeating the word "helpful" 200 times
- Always agreeing with the user (sycophancy)
- Generating gibberish that happens to have tokens the RM likes
- Producing very long answers because longer = "more thorough" to the RM

Why this happens

The RM is imperfect. If we optimize *too hard* against an imperfect signal, we exploit its flaws.

The Fix: Stay Close to the SFT Model

The full RLHF objective

$$\max_{\theta} \; \mathbb{E}\big[r_\phi(x, y)\big] \; - \; \beta \cdot \mathrm{KL}\big(\pi_\theta \,\|\, \pi_{\mathsf{SFT}}\big)$$

Plain English:

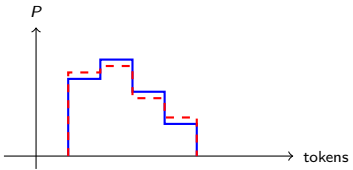
- First term: get a high reward
- Second term: **penalty** for drifting too far from the SFT model
- β controls the trade-off

In one sentence

"Be better, but don't become weird."

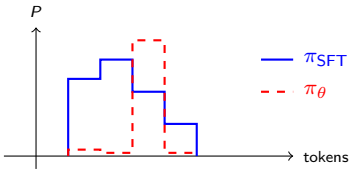
What Is KL Divergence? (Visual)

KL divergence measures how different two probability distributions are.



Low KL (similar)

$\pi_\theta \approx \pi_{\text{SFT}}$ — OK



High KL (very different)

π_θ has drifted — penalize!

The KL penalty **keeps the new policy close to the original SFT model**—preventing weird, unnatural outputs.

PPO: The Optimization Algorithm

To actually optimize this objective, InstructGPT uses **PPO** (Proximal Policy Optimization).

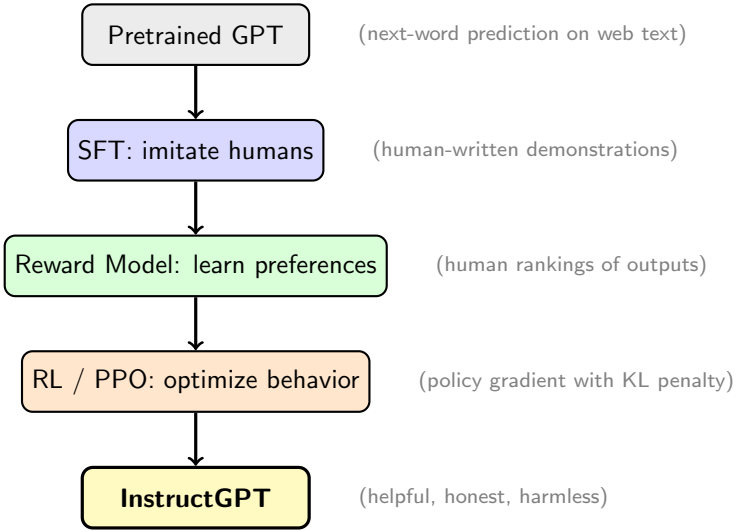
PPO in one paragraph

PPO is a reinforcement learning algorithm that updates the policy in **small, stable steps**. It avoids making any one update so large that the policy becomes unstable. This is important when your "reward" is itself a learned model that could be exploited.

- You don't need to know the internals for this course
- Just know: **it's a stable way to do gradient updates in RL**
- Alternatives exist (DPO, REINFORCE, RLOO...), but PPO was the original choice

"PPO is the gradient-descent of RLHF."

The Full RLHF Pipeline



Key Results from the Paper

The headline result

A **1.3B** InstructGPT is preferred by humans over a **175B** GPT-3
(that's a model **100**$\times$ **smaller** winning on user preference).

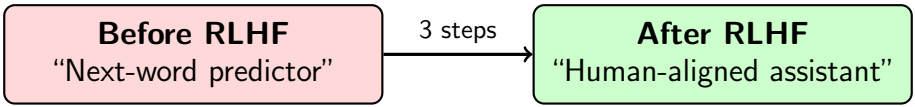
Specifically, InstructGPT is:

- Better at following instructions
- Less likely to make up facts (reduced hallucination)
- More truthful on TruthfulQA
- Less toxic on safety benchmarks
- Slightly worse on some NLP benchmarks ("alignment tax")

The lesson

Alignment matters more than raw size.
Good data + good training procedure beats brute-force scaling.

Final Intuition: What Changed?



The one-sentence summary

RLHF takes a model that knows **how to generate text** and teaches it **how to be useful**.

- Step 1 (SFT): show it what good looks like
- Step 2 (RM): learn a scorer from human comparisons
- Step 3 (RL): optimize against that scorer, carefully

Questions to Think About

1. If the reward model is biased, what happens to the final policy?
2. Why not just train more humans to write better SFT data? (Hint: consider scale and cost.)
3. What happens if β (the KL weight) is too small? Too large?
4. Could we replace human rankings with AI rankings? (Yes—this is "RLAIF.")
5. Where does Constitutional AI / DPO fit in this picture?

Reference

Ouyang et al. (2022). "Training language models to follow instructions with human feedback." arXiv:2203.02155.