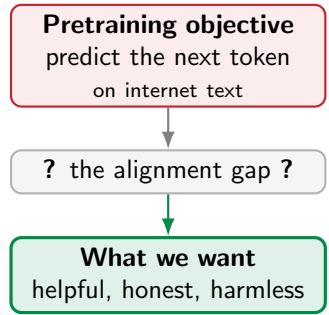
Your model does not do what you ask

A pretrained LM is trained on one objective only: **predict the next token** on a web page. Prompt GPT-3 with an instruction and you get:

- **Prompt:** "Explain the moon landing to a 6 year old."
- **GPT-3:** "Explain the theory of gravity to a 6 year old. Explain the big bang theory to a 6 year old. Explain evolution..."

It just *continues the pattern*. It is not being unhelpful on purpose – following instructions was never its training objective.

Alignment is the gap between "predict the next token" and "do what the user meant." Today: the recipe – **RLHF** – that first closed it at scale.



Outline

Why a bigger model is not a better assistant

The training objective and the goal are two different things.

Next-token prediction $\neq$ following instructions

Scaling up the language-modelling objective makes models more *fluent*, not more *obedient*. The same 175B model that writes beautiful prose will happily:

- make up facts that sound authoritative (**hallucinate**),
- produce biased or **toxic** text,
- ignore an explicit constraint, or answer a different question entirely.

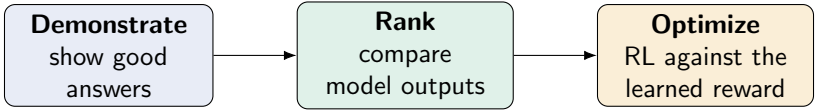
The fix is not more data or more parameters – it is a **different objective**. We want to optimize for what Askell et al. (2021) call the three H's:

Helpful (solve the user's task) · **Honest** (do not fabricate) · **Harmless** (do no harm)

The problem: "helpful, honest, harmless" has **no loss function**. You cannot write it down and differentiate it. So we *learn* it – from human judgement.

The idea: turn human preference into a reward

We cannot specify good behaviour, but a human can **recognize** it: shown two answers, they can say which is better. RLHF turns that ability into a training signal.



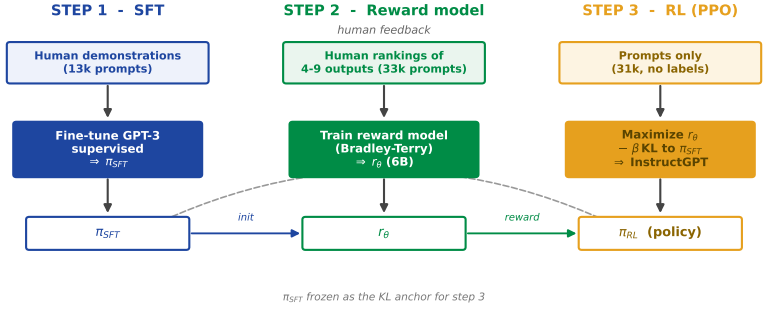
This is the three-step pipeline of InstructGPT – **SFT**, a **reward model**, and **PPO**. It is the *foundational* alignment recipe: later methods (GRPO, DPO) are best understood as *simplifications* of exactly this pipeline.

The three-step pipeline

SFT $\rightarrow$ reward model $\rightarrow$ PPO. Learn each stage once.

The whole recipe on one slide

RLHF: supervised demos → a learned reward → RL against it



Each stage consumes a **different kind of human input**: demonstrations (step 1), rankings (step 2), and then *no* labels at all in step 3 – the reward model stands in for the human.

Step 1: supervised fine-tuning (SFT)

Labelers **write the ideal response** to each prompt. Fine-tune GPT-3 on these demonstrations with ordinary supervised learning (maximize log-likelihood of the target):

$$\max_{\theta} \; \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{demo}}} \sum_{t} \log \pi_\theta(y_t \mid x, y_{<t})$$

- $\sim$13k demonstration prompts, 16 epochs.
- Teaches the *format*: "respond to instructions" instead of "continue text."

SFT alone already helps a lot – but it can only imitate the answers labelers *wrote*. It cannot learn from answers the *model* produced that turned out good or bad.

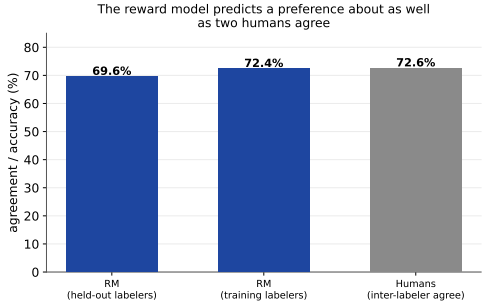
That is exactly what steps 2–3 add: judge the model's *own* outputs.

Step 2: a reward model from rankings

Show a labeler $K = 4$–9 model outputs for one prompt; they **rank** them. Each pair $(y_w \succ y_l)$ is a training example for a reward model $r_\theta(x, y)$, via the **Bradley–Terry** pairwise loss:

$$\mathcal{L}(\theta) = -\,\mathbb{E}_{(x,y_w,y_l)}\Big[\log\ \sigma\big(r_\theta(x,y_w) - r_\theta(x,y_l)\big)\Big]$$

- Output: one **scalar** reward per (prompt, answer).
- Rank, don't score: humans compare reliably, absolute scores drift.
- A single 6B RM (the 175B RM was unstable).



The RM predicts a held-out preference nearly as well as two humans agree with each other – a genuine, if imperfect, stand-in for the labeler.

Step 3: optimize the policy with PPO

Now treat generation as a one-step RL problem: the prompt is the state, the whole answer is the action, and the reward model scores it. Fine-tune the SFT policy to **maximize reward**, with a **KL penalty** that keeps it near the SFT model:

$$\text{objective}(\phi) = \mathbb{E}_{(x,y)\sim\pi_\phi^{RL}}\Big[\underbrace{r_\theta(x,y)}_{\text{reward}} - \beta\underbrace{\log\frac{\pi_\phi^{RL}(y|x)}{\pi^{SFT}(y|x)}}_{\text{KL to SFT}}\Big] + \gamma\underbrace{\mathbb{E}_{x\sim\mathcal{D}_{\text{pretrain}}}\big[\log\pi_\phi^{RL}(x)\big]}_{\text{pretraining mix (ptx)}}$$

- **KL penalty** (β): stops the policy from drifting into gibberish that games the reward model (reward hacking).
- **Pretraining mix** (γ): the "ptx" term – fixes regressions on public NLP tasks (more on this in results). Plain PPO sets $\gamma = 0$.

Everything here rides on **PPO**. Next section: what PPO actually is, and why it clips.

A PPO primer

Just enough reinforcement learning to understand step 3.

Why not just ascend the reward?

The policy-gradient update pushes up the probability of actions with positive **advantage** A_t (better than expected) and down for negative:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J} = \mathbb{E}_t\big[\, A_t \, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t \mid s_t)\big]$$

- We estimate A_t from samples, so it is **noisy**.
- A big step in a noisy direction can **destroy** the policy – and unlike supervised learning, the next batch is generated *by that broken policy*. No recovery.

We want the biggest improving step that still **stays close** to the current policy – a **trust region**. PPO enforces one cheaply, with a clip.

Why a ratio at all? The data is off-policy

A subtlety hides in step 3. You sample a batch of answers from the **current** policy, score them, then take *several* gradient steps on that one batch. After the first step the policy has *moved* – so the later steps are learning from data generated by an **older** version of themselves.

- To reuse old-policy data honestly you must **reweight** each action by how likely the *new* policy is to take it – that reweighting is the **importance ratio** $r = \pi_{\text{new}}/\pi_{\text{old}}$.
- But the reweighting is only trustworthy while the two policies stay *close*. Let them drift far apart and the estimate blows up.

So we need two things at once: **reuse** the batch (the ratio) and **stay close** so the reuse stays valid (the clip). The next slide is exactly those two ideas in one line.

The importance ratio and the clipped surrogate

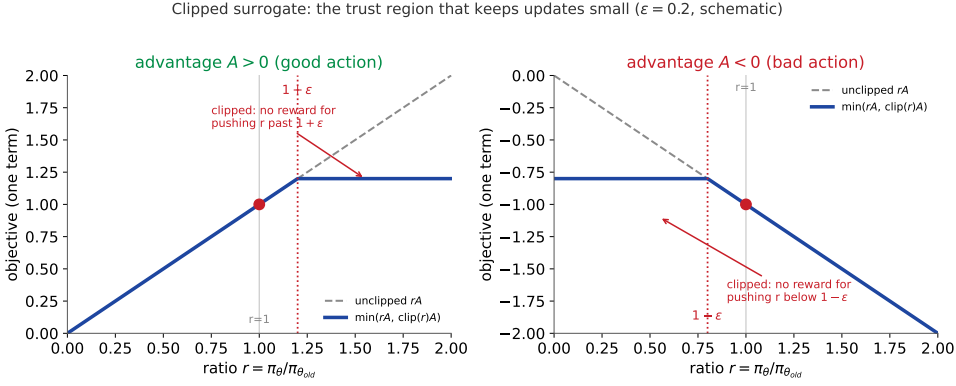
Let $r(\theta) = \dfrac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{old}}(a_t \mid s_t)}$ be the **importance ratio**: how much more likely the new policy makes an action the old one took. PPO maximizes the **clipped surrogate**:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t\Big[\min\big(r(\theta)\,A_t,\ \text{clip}(r(\theta),\,1-\varepsilon,\,1+\varepsilon)\,A_t\big)\Big]$$

- $r = 1$ at the start (new = old policy).
- The min makes it a **pessimistic lower bound**: you only get credit for moving r inside $[1-\varepsilon, 1+\varepsilon]$.
- Typical $\varepsilon = 0.2$.

Past the clip edge the objective goes **flat**: zero gradient, so no incentive to keep moving the ratio. That is the trust region.

Why clipping keeps you in the trust region



- **Good action ($A > 0$):** increasing r helps – but only up to $1 + \varepsilon$, then it flattens. No reward for over-committing.
- **Bad action ($A < 0$):** decreasing r helps – but only down to $1 - \varepsilon$. The clip removes the incentive for a giant destructive step in either direction.

Where does A_t come from? The critic

The advantage needs a **baseline**: "better than *what*?" PPO learns a second network, the **value model** (critic) $V_\psi(s)$, predicting expected reward, and sets

$$A_t \approx r_\theta - V_\psi(s) \quad \text{(via GAE)}.$$

Subtracting a baseline cuts gradient *variance* without adding bias.

In InstructGPT the critic is **initialized from the reward model**. So RLHF-PPO keeps several models live at once.

PPO for RLHF juggles **four** models:

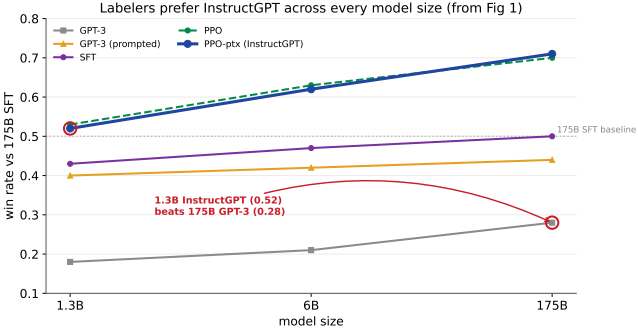
1. Policy π_ϕ (trained)
2. Reference π^{SFT} (frozen, KL anchor)
3. Reward r_θ (frozen)
4. **Value V_ψ (trained, big)**

Remember the critic – the later decks delete it.

Does it work?

The headline result, and the honesty / harmlessness numbers.

The headline: 1.3B beats 175B



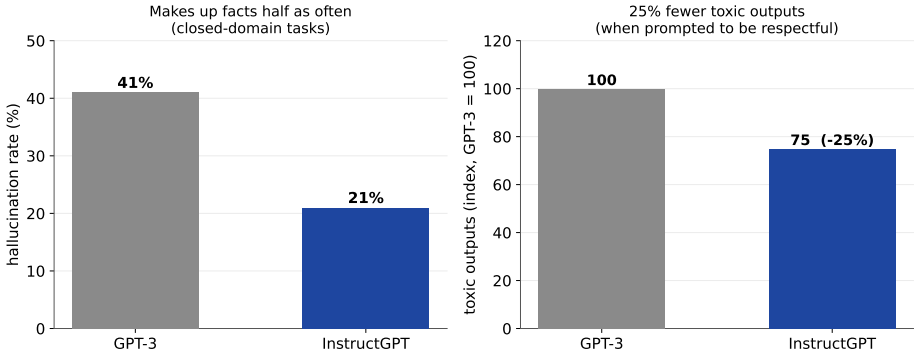
Win rate = how often labelers prefer a model's output to the 175B SFT baseline.

The striking result: the **1.3B** InstructGPT is preferred to the **175B** GPT-3 – a model **100**× larger.

Head to head, 175B InstructGPT beats 175B GPT-3 **85%** of the time, and beats few-shot-prompted GPT-3 **71%** of the time.

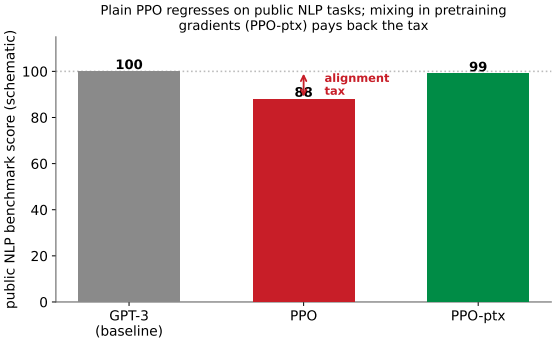
Alignment bought more preference than a 100× scale-up.

Honesty and harmlessness improve too



- **Truthfulness:** on TruthfulQA, InstructGPT is truthful *and* informative about **twice** as often as GPT-3 – and it does this by default, without being told to.
- **Honesty / harm:** it hallucinates half as often on closed-domain tasks (21% vs 41%) and, when asked to be respectful, emits **25%** fewer toxic outputs.

The alignment tax – and how to pay it back



Plain PPO buys alignment at a price: it **regresses** on public NLP benchmarks (SQuAD, DROP, HellaSwag, translation). That gap is the **alignment tax**.

Mixing pretraining gradients back into PPO (the **ptx** term, $\gamma > 0$) recovers almost all of it *without* hurting labeler preference – and beats simply raising the KL coefficient.

A high alignment tax is dangerous: it tempts people to deploy the *unaligned* but more capable model. Low-tax alignment is the goal.

It generalizes beyond the training signal

Two encouraging kinds of generalization – alignment "sticks" where we never supervised it:

- **Held-out labelers.** Workers who produced *none* of the training data prefer InstructGPT over GPT-3 at about the same rate – so it is not just overfitting to the 40 training labelers.
- **Held-out instruction types.** Though the fine-tuning data is >96% English and barely any code, InstructGPT follows instructions in **other languages** and **answers questions about code** – GPT-3 needs careful prompting and usually will not.

The model seems to learn the *notion* of "follow the instruction," not just the specific instructions it was trained on. Alignment is cheap, too: the 175B RLHF run was ∼60 petaflop/s-days vs 3,640 to pretrain GPT-3.

Limitations, and why this pipeline got simplified

A foundational recipe – and a heavy one.

InstructGPT is aligned, not solved

- **Still makes simple mistakes:** follows *false premises*, over-hedges simple questions, and breaks under multiple explicit constraints.
- **Still not safe:** it will follow a *harmful* instruction; told to be maximally biased, it is *more* toxic than GPT-3. It optimizes helpfulness, and that can conflict with harmlessness.
- **Aligned to whom?** To $\sim$40 mostly English-speaking contractors and the researchers' instructions – not "humanity." Labelers agree only $\sim$73% of the time.

And the *machinery* is heavy: a separate reward model, an RL loop, and PPO's extra **value network** – up to four models in memory. That cost is exactly what the next papers attack.

Recap: the pipeline everything else simplifies

- Pretraining optimizes next-token prediction; **alignment** is the gap to "helpful, honest, harmless," and RLHF closes it by **learning a reward from human preference**.
- **Three steps:** SFT on demonstrations $\rightarrow$ a Bradley–Terry **reward model** from rankings $\rightarrow$ **PPO** against that reward with a KL penalty to SFT.
- **PPO** keeps updates in a trust region via the clipped surrogate, and needs a **value network** (critic) for the advantage baseline.
- **Results:** 1.3B InstructGPT $\succ$ 175B GPT-3; more truthful, less toxic; the alignment tax is paid back by the pretraining mix.

Next: the field spent years *deleting pieces of this pipeline.* *GRPO* (**[LLM-1]**) throws out PPO's value network, replacing the critic with a group-relative baseline. *DPO* (**[LLM-2]**) throws out the reward model *and* the RL loop entirely, folding preference optimization into one classification-style loss.