RL and RLHF

🧠 Reinforcement Learning (RL) — Detailed Explanation

1️⃣ What is Reinforcement Learning?

Reinforcement Learning (RL) is a branch of machine learning where an **agent** learns to make **sequential decisions** in an **environment** to maximize some **notion of cumulative reward**.

Unlike supervised learning (where we learn from labeled examples), in RL:

- The agent **learns by interacting** with the environment.
- It **receives feedback** (rewards or penalties) based on its actions.
- Over time, it **learns an optimal policy** — a strategy for choosing actions that yield the highest long-term reward.

💡 Analogy:

Think of training a dog:

- **Agent** → the dog
- **Environment** → the world it's in
- **Action** → what it does (sit, run, fetch)
- **Reward** → treat for good behavior
- **Policy** → it's a learned behavior strategy over time

2️⃣ Core Components of Reinforcement Learning

RL is formally described using a **Markov Decision Process (MDP)**, defined by a 5-tuple:

$$\text{MDP} = (S, A, P, R, \gamma)$$

Markov Decision Process (MDP) — Table Summary

Symbol	Component	Description
S	States	All possible situations the agent can be in
A	Actions	Set of possible actions the agent can take
P	Transition Probability	$P(s' \mid s, a)$
R	Reward Function	(R(s, a)): Immediate feedback after action (a) in state (s)
γ (gamma)	Discount Factor	Controls how much future rewards are valued compared to immediate rewards

3️⃣ The RL Loop (Agent–Environment Interaction)

At each time step t:

1. The agent **observes** the current state s_t.
2. It **chooses an action** a_t according to its **policy** $\pi(a_t|s_t)$.
3. The environment **responds**:
   - Returns a **reward** r_t.
   - Transitions to a **new state** s_{t+1}.
4. The agent **updates its knowledge or policy** based on the experience (s_t, a_t, r_t, s_{t+1}).

This cycle continues until the task (episode) ends.

4️⃣ Policy — The Agent's Decision Strategy

📘 Definition:

A **policy** π defines the agent's behavior — it is a mapping from states to actions:

$$\pi(a|s) = P(a_t = a \mid s_t = s)$$

That is, the **probability of taking action a** when in state s.

💡 Types of Policies:

Type	Description
Deterministic Policy	Always picks the same action for a given state , $a = \pi(s)$
Stochastic Policy	Chooses actions according to a probability distribution $\pi(a \mid s)$

The policy is **parameterized by θ**, giving us $\pi_\theta(a|s_t)$, where **θ** represents the learnable parameters of the model (like weights in a neural network).

5️⃣ Objective of RL

The main goal is to find an **optimal policy** π^* that **maximizes the expected cumulative reward** (called the *return*).

$$J(\theta) = \mathbb{E}\pi\theta \left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

Where:

- r_t = reward at time t
- $\gamma \in [0, 1]$ = discount factor
- The expectation is taken over trajectories following policy π_θ

✅ Intuition:

We want to learn how to act such that, in the long run, total reward is maximized — not just immediate gratification.

6️⃣ Value Functions

Value functions estimate *how good* it is to be in a certain state or take a certain action.

Function	Definition	Meaning
State-Value Function $V^\pi(s)$	$\mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s]$	Expected return from state s following policy π
Action-Value Function $Q^\pi(s,a)$	$\mathbb{E}_\pi[\sum_t \gamma^t r_t \mid s_0 = s, a_0 = a]$	Expected return from taking action a in state s

These functions guide the agent toward high-reward decisions.

7 Major Categories of RL Algorithms

Type	Examples	Key Idea
Value-based	Q-Learning, Deep Q-Network (DQN)	Learn a value function (Q(s,a)) and pick the action with highest Q-value
Policy-based	REINFORCE, PPO, A2C	Directly learn a parameterized policy \pi_\theta(a
Actor–Critic	A3C, PPO, DDPG	Combine both: Actor updates the policy, Critic estimates the value function
Model-based	MuZero, AlphaZero	Learn an internal model of environment dynamics

8 Policy Gradient Methods

In **policy-based RL**, we directly optimize the parameters θ of the policy $\pi_\theta(a|s)$ using gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}\pi\theta[\nabla_\theta \log \pi_\theta(a|s), Q^\pi(s,a)]$$

Intuition:

- Encourage actions that yield high Q-values.
- Discourage actions that yield low Q-values.
- The model updates θ in the direction that increases the expected reward.

9 Exploration vs Exploitation Dilemma

- **Exploration:** Trying new actions to discover better strategies.
- **Exploitation:** Using known actions that yield high rewards.

A good RL agent **balances** both:

- Too much exploration → slow convergence
- Too much exploitation → stuck in local optima

Algorithms like **ε-greedy** or **Boltzmann exploration** help balance this.

🔟 Applications of Reinforcement Learning

Domain	Application
Games	AlphaGo, AlphaZero, OpenAI Five
Robotics	Path planning, control of robotic arms
Recommendation Systems	Personalized content delivery
Finance	Portfolio optimization, algorithmic trading
Autonomous Driving	Decision-making and path control
Healthcare	Adaptive treatment, drug discovery

🧾 Key Takeaways

- RL is about **learning by interaction**, not labeled data.
- The agent learns a **policy** $\pi_\theta(a|s_t)$ — a probability distribution of actions given the current state.
- The goal is to **maximize expected cumulative reward**.
- RL methods are categorized into **value-based**, **policy-based**, and **actor-critic**.
- RL underpins many **modern AI systems** — from AlphaGo to LLM alignment via RLHF.

🧩 In One Line:

> Reinforcement Learning is the science of training an agent to act optimally in an environment by learning from feedback, aiming to maximize long-term reward through a policy $\pi_\theta(a|s_t)$.

🧠 Architecture of a Reward Model (RM)

A **Reward Model** is a neural network trained to **score** model outputs based on human preferences.

It converts **qualitative human judgments** (e.g., "Response A is better than B") into a **scalar reward signal** that an RL algorithm can optimize.

1️⃣ Overall Structure

A Reward Model is typically built by taking a **pretrained LLM encoder** and adding a **simple head** on top:

```
[Input: Prompt + Candidate Response]
            |
            ▼
    Pretrained LLM (e.g., GPT, T5)
            |   → transforms text into hidden states
            ▼
    Reward Head (usually 1 linear neuron)
            |
            ▼
    Scalar Reward Score (e.g., 1.7)
```

2️⃣ Detailed Components

a) Input Format

- The input to the RM is the **prompt concatenated with a candidate answer**.
- Example:

> "Q: What is photosynthesis?
> A: It is the process by which plants create food..."

b) Backbone Model

- A pretrained transformer (same architecture as the base LLM) processes the entire sequence.
- Output: hidden state for each token.

c) Reward Head

- A **single linear layer** maps the final hidden state (or pooled representation) to a scalar:

$$r_\theta(x, y) = w^\top h_{\text{end}}$$

- No softmax — reward is a **raw score**, not a probability.

d) Output

- A single **continuous value** representing how "good" the response is.
- Higher score = better response (according to human preference).

3️⃣ Training Objective of the Reward Model

The RM is trained using **pairwise preference data** collected from humans.

Given two candidate responses A and B for the same prompt:

- Human says: *A is better than B*
- The reward model should output:

  r(A) > r(B)

Pairwise Ranking Loss

$$L = -\log \sigma(r_\theta(A) - r_\theta(B))$$

This loss:

- Increases when A is rated better than B
- Punishes the model if the scores disagree with human preferences

---

4️⃣ Why Use a Reward Model?

- The RL algorithm (like PPO) needs a **numeric reward**.
- Humans can't write numeric rewards directly.
- The reward model becomes a **surrogate for human judgment**.

5️⃣ Key Characteristics

Feature	Description
Built on top of an LLM	Uses same transformer backbone as base model
Scalar output	Just one number per prompt–response pair
Trained with preference ranking	A > B comparisons, not labels
No task-specific labels	Only human preference signals
Used in RLHF	Guides policy optimization

6️⃣ Where It Fits in RLHF Pipeline

1. **Supervised Fine-Tuning (SFT)**

   Model learns basic helpful behavior.

2. **Reward Model Training**

   Human comparisons → Train RM to score outputs.

3. **Reinforcement Learning (e.g., PPO)**

   Policy optimized to maximize RM's reward.

This produces aligned, safe, higher-quality responses.

🧾 One-Line Summary

The reward model is a pretrained LLM with a small linear head on top that outputs a scalar score trained on human preference comparisons, enabling RL algorithms to optimize model behavior according to human values.

🧠 Reinforcement Learning from Human Feedback (RLHF) — Detailed Explanation

RLHF is the method used to align large language models (LLMs) like ChatGPT with **human preferences, values, and instructions**.

Instead of relying only on text data from the internet, the model is additionally trained using **human feedback** about what good outputs look like.

It is a **3-stage pipeline**:

1️⃣ **Supervised Fine-Tuning (SFT)**

2️⃣ **Reward Model (RM) Training**

3️⃣ **Reinforcement Learning (PPO) using the Reward Model**

Let's break down each stage clearly.

1️⃣ Stage 1 — Supervised Fine-Tuning (SFT)

Purpose:

Give the model an initial understanding of *how* to follow instructions.

Process:

- Human annotators write high-quality **instruction → response** examples.
- The pretrained LLM is fine-tuned on this dataset using **supervised learning**.

Example:

```
User: Explain photosynthesis.
Assistant: Photosynthesis is the process....
```

The model becomes much better at:

- Following instructions
- Answering in human-preferred styles
- Providing helpful and safe content

But it still doesn't perfectly match human preferences → we need more.

2️⃣ Stage 2 — Reward Model (RM) Training

After SFT, the model can generate multiple possible responses — but which ones humans prefer?

We train a **Reward Model** to learn these preferences.

How it works:

1. Take the SFT model.
2. For a single prompt, generate **multiple candidate responses** (e.g., 4 outputs).
3. Human annotators **rank them** from best → worst.
4. Train a neural model (R_ϕ) so that:
    - Higher-ranked responses get **higher scores**
    - Lower-ranked responses get **lower scores**

Reward Model Objective:

For a pair (preferred response (y^+) and dispreferred (y^-)):

$$R_\phi(y^+) > R_\phi(y^-)$$

The RM learns:

- What humans find helpful
- What is harmful or undesirable
- What tone/style is preferred
- What answers are good vs hallucinated

This RM becomes a **learned human preference function**.

3️⃣ Stage 3 — Reinforcement Learning (PPO) Using RM

Now we use **PPO (Proximal Policy Optimization)** — a stable RL algorithm — to fine-tune the LLM **to maximize the reward model score**.

The policy is the LLM:

$\pi_\theta(a|s)$

Where:

- (s) = text prompt + previous tokens
- (a) = next token chosen

The LLM is trained to:

- Generate responses that the Reward Model **scores highly**
- Avoid responses, RM scores low

PPO Loss:

PPO modifies the LLM so that:

- It **improves** (more reward)
- But **doesn't change too drastically** from the SFT model

  (to avoid “reward hacking” or extreme outputs)

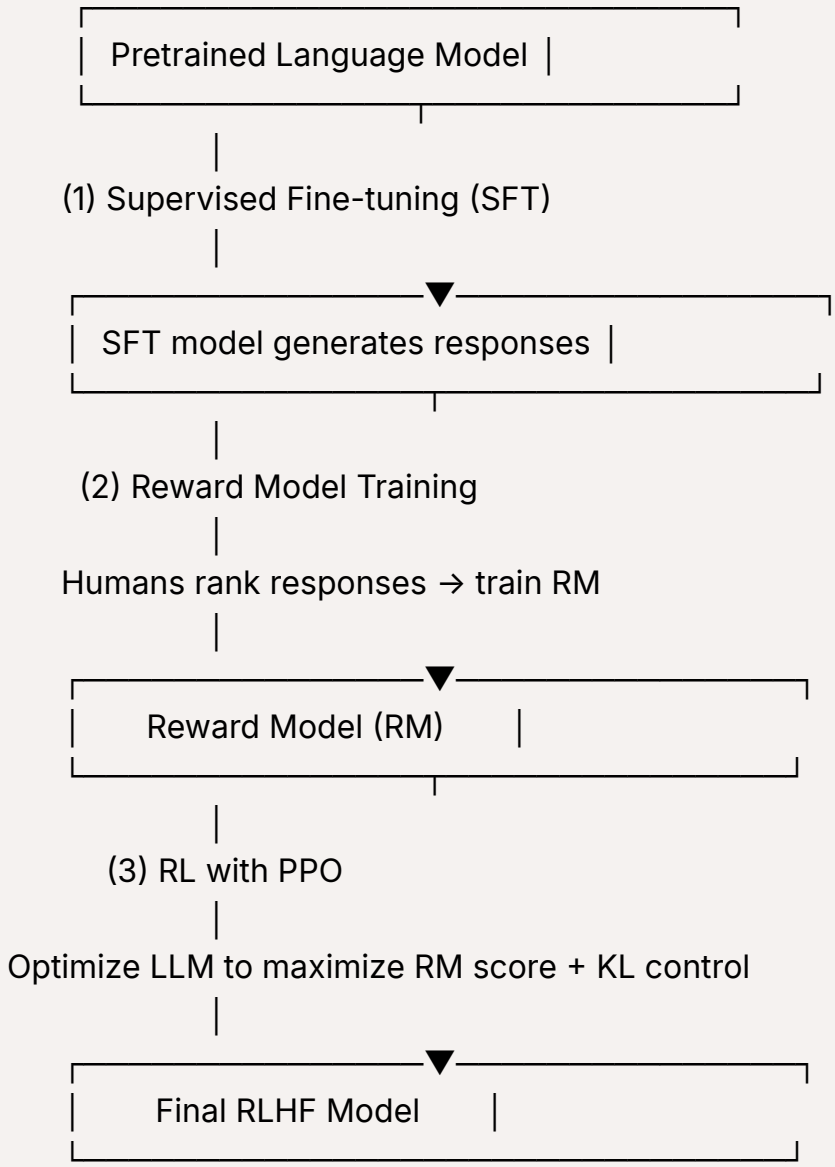
The PPO objective includes:

- **policy gradient term** (maximize reward)
- **KL penalty** toward the SFT model (to prevent divergence)

Final Result:

The aligned policy (π^*) creates - Helpful, Harmless, Truthful, Human-preferred content.

🔁 Full RLHF Pipeline Summary



🎯 Why RLHF Works Well

✨ Better alignment

Models learn what **humans actually want**, not just what internet text looks like.

✨ Safety & Filtering

Bad or harmful outputs get low reward → the model avoids them.

✨ Follows instructions better

RLHF improves:

- Cooperation
- Helpfulness
- Step-by-step reasoning
- Conversation quality

This is why ChatGPT feels so “aligned” compared to raw LLMs.

⚠️ Limitations of RLHF

- Reward Model can learn **incorrect preferences**
- PPO optimization can cause **over-optimization** (reward hacking)
- Some tasks require multi-step reasoning that RM cannot evaluate well
- Quality depends heavily on human annotators
- RM only reflects “average human preference,” not universal truth

Modern systems often combine RLHF with:

- Constitutional AI
- Self-alignment
- Tool-use
- Retrieval augmentation
- Multi-agent feedback

 In One Line

> RLHF aligns a pretrained language model with human preferences using a 3-step pipeline: supervised fine-tuning, reward model training, and reinforcement learning with PPO.

🔴 Combining the Objective: Regularized Reward Maximization

In RLHF, we want to train a language model to produce outputs that:

1. **Get high reward** (according to a learned reward model)
2. **Do not drift too far** from the original pretrained model's behavior

So the final objective must balance **reward maximization** with **regularization**.

This slide visually shows how these two pieces combine.

1️⃣ Maximize the Reward

Goal: make the model output responses (y) that have **high reward** (r(x, y)).

Mathematically:

$$\mathbb{E}y \sim \pi\theta(y|x)[r(x, y)]$$

- (x) = input prompt
- (y) = generated response
- (π_θ) = the policy (the RL-trained model)
- Reward model assigns a scalar reward to each output.

👉 **This pushes the model to generate the "best" answers according to human feedback.**

2️⃣ Minimize KL Divergence (Stay Close to Base Model)

If you only maximize reward, the model may:

- Over-optimize reward
- Start generating unnatural, verbose, or repetitive text
- Drift far from the pretrained LLM
- Exhibit "reward hacking"

So we add a **penalty** to keep (π_θ) close to the **base policy** (π_{ref}) (the original pretrained model).

This penalty is:

$\mathrm{KL}(\pi_\theta, ||, \pi_{\text{ref}})$

Expanded inside expectation:

$$\mathbb{E}y \sim \pi\theta(y|x) \left[\log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}\right]$$

👉 **This prevents the model from changing too much from its original safe behavior.**

The slide shows that this KL term goes **downwards**, meaning we want to **minimize** it.

3️⃣ Add a Scaling Factor β (or λ) to Control the Balance

We need to balance:

- **How much reward to maximize**
- **How much KL to penalize**

So we multiply KL by a constant (\lambda) (sometimes written as β in papers).

Final combined objective:

$$\mathbb{E}_{\pi_\theta(y|x)} \left[r(x,y) - \lambda \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}\right]$$

This is exactly what the bottom equation in the slide shows.

4️⃣ Interpretation: Why This Form?

💡 This is a regularized optimization problem:

- **Reward term** → Move toward preferred outputs
- **KL term** → Keep model safe, stable, and human-like

Equivalent interpretation:

The model tries to choose (y) such that:

- **High reward**
- **Small deviation** from the base LLM

Intuition:

Think of it like stretching a rubber band:

- Reward pulls the model toward new, better behaviors
- KL is the tension holding the model near where it was trained

---

5️⃣ Why This Works in RLHF

This objective ensures:

1. The model **improves** by learning human preferences
2. But does **not** diverge into unsafe or strange responses
3. Training is stable (KL prevents runaway gradient updates)
4. Allows the model to stay strongly grounded in pretrained knowledge

It is the core reason RLHF works well on large LLMs.

✅ Summary (for notes)

- RLHF final objective = **reward maximization + KL penalty**
- Reward encourages good answers
- KL divergence prevents drifting too far from original LLM
- Scaling factor (\lambda) adjusts how conservative or aggressive the model is
- Combined objective:

$$\mathbb{E}_{\pi_\theta(y|x)}\left[r(x,y) - \lambda \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}\right]$$

This is the **PPO-style regularized reward objective** widely used in ChatGPT-like RLHF training.
