COMS4995W32
Applied Machine Learning

Dr. Spencer W. Luo

Columbia University | Fall 2025

Reinforcement Learning in LLMs

Agenda

- Motivation
- Data for RL
- RL Algorithms for LLMs
- RL for Thinking Improvements

Motivation

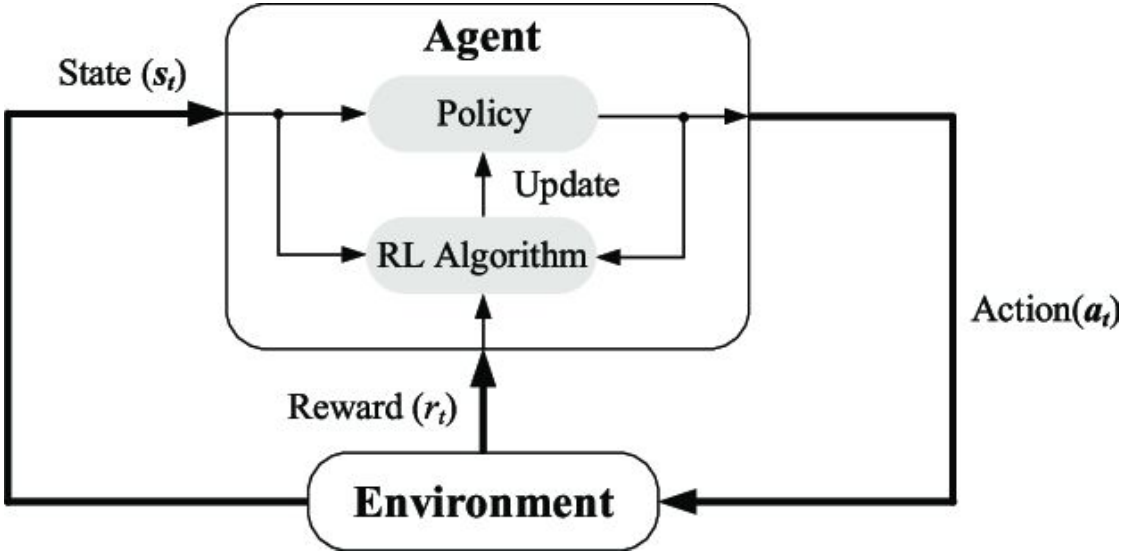
What is Reinforcement Learning (RL)? 🧭

Reinforcement Learning (RL) is a paradigm where an agent interacts with an environment, takes actions, and receives rewards; over time it learns a policy model that **maximizes expected cumulative reward**

Unlike SFT (imitate labeled examples), RL learns by doing

- tries things
- sees consequences
- updates behavior accordingly

What is Reinforcement Learning (RL)? 🧭



What is Reinforcement Learning (RL)? 🧭

Core loop:

1️⃣ The agent observes a state `s`

2️⃣ Chooses an action `a` based on its policy `π(a|s)`

3️⃣ Receives a reward `r` and new state `s'`

4️⃣ Updates its policy `π` to maximize expected future rewards

What is Reinforcement Learning (RL)? 🧭

Goal: Find a policy π* that maximizes

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

where γ is the discount factor (balances short-term vs long-term gains)

Robot Example 🤖

A robot is learning to navigate a maze

- r_t = immediate feedback the agent receives at time t
- At the goal: r_t = +1 → a strong positive signal for success
- Each step: r_t = −0.01 → a small penalty encouraging efficiency
- Goal: reach the goal quickly (avoid many −0.01 steps)

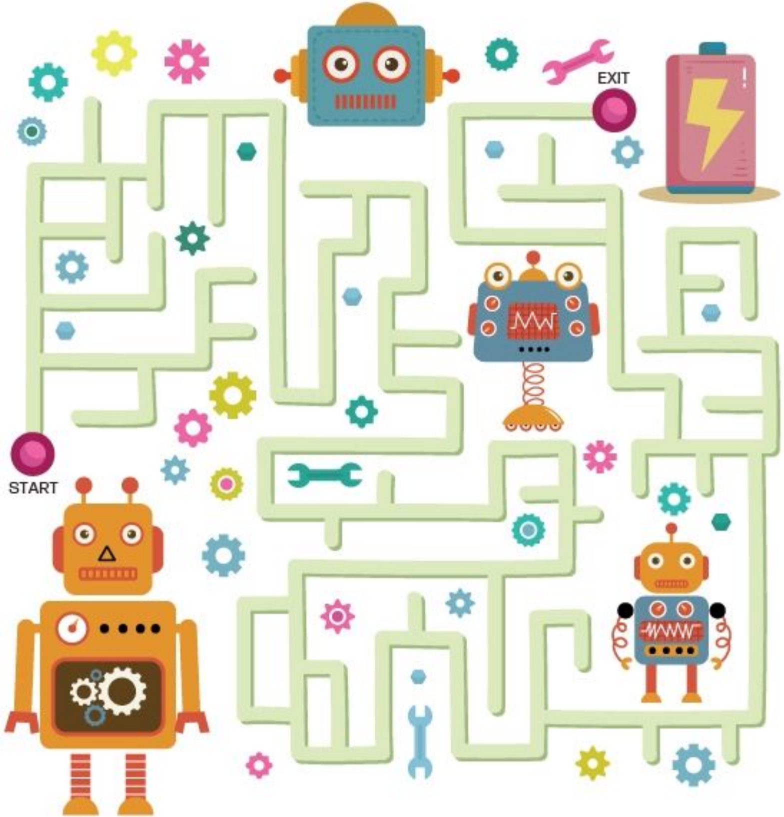
$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

γ=0.9 → earlier rewards still valuable

γ=0.1 → only immediate rewards count

ROBOT MAZE

Help Mr. Robot reach the battery.



Why RL for LLMs? 🚀

LLMs first learn by imitation:

- Pre-training: predict the next token from massive text corpora
- Supervised Fine-Tuning: imitate high-quality human answers

Both stages teach models to sound right, not necessarily be right

They imitate human patterns but do not optimize for deeper goals

Reinforcement Learning introduces explicit objectives through rewards

→ to optimize for quality, usefulness, safety and more

What Objectives is LLM Optimizing For? 🧭

Helpfulness - useful, relevant, stays on task, concise when needed

Honesty - factually grounded, transparent about uncertainty

Harmlessness - avoids unsafe, biased, or toxic content

Reasoning - correct intermediate logic, verifiable steps

Efficiency - answers quickly and clearly, minimal redundancy

Creativity - generates diverse, novel yet relevant ideas

How RL Maps to LLMs 🤖

Agent = the language model π itself

It observes a prompt (state) and decides what to say next

Action = each generates a series of tokens

Each token choice changes the conversation or reasoning trajectory

Environment = the task context

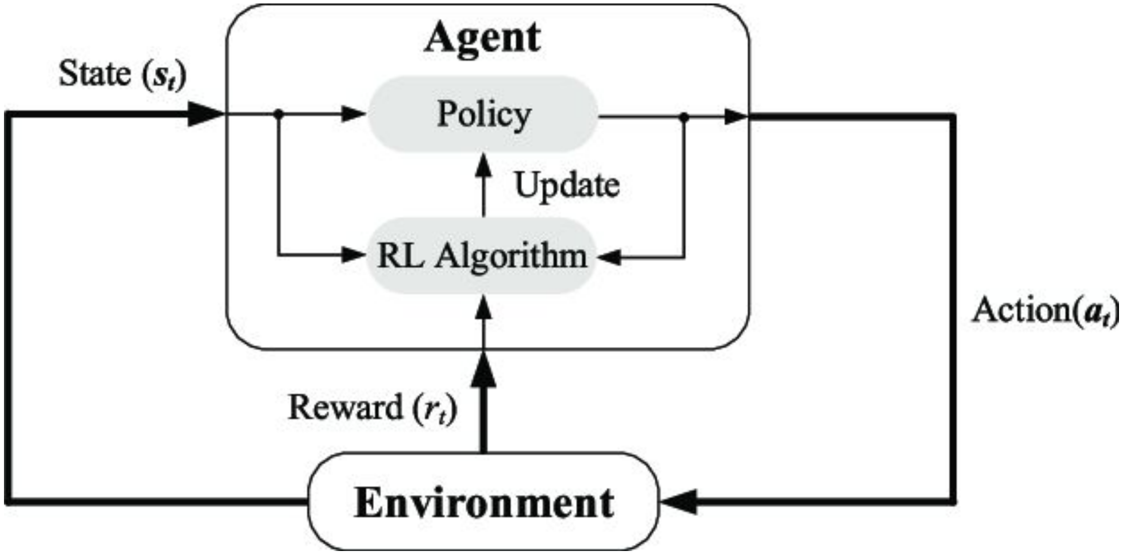
It provides the prompt, previous dialogue, or code problem

Reward = feedback signal

Comes from humans (preferences), AI judges, or verifiers (pass/fail)

ChatGPT is like an RL agent taking a good action and receiving a high reward

What is Reinforcement Learning (RL)? 🧭



Three Reward Regimes 📊

Preference-based (RLHF)

- Humans compare two responses → label A > B
- Captures subjective qualities like clarity, politeness, helpfulness
- Costly but grounds the model in human intent

AI-based (RLAIF)

- A stronger model or rule-based constitution acts as judge
- Scalable, fast, consistent
- Risk: feedback bias or “echo” - model rewards its own habits

Three Reward Regimes 📊

Verifiable (Correctness)

- External checker or test provides pass/fail or numeric score
  - unit tests for code, math solver for proofs, KB for facts
- Dense and unbiased signals → enables self-improvement loops

Recent Trend (2024 -):

Research and industry move toward verifiable rewards as the foundation for reliable reasoning and self-improvements

High-Level Pipeline 🧰

1️⃣ Collect Data & Rewards

Gather prompts, outputs, and feedback signals - human, AI, or verifier

2️⃣ Train Reward / Verification Mechanism

Convert feedback into a reward model or objective checker R(x,y)

3️⃣ Optimize Policy (PPO / DPO / variants)

Update model to maximize reward while staying close to reference

4️⃣ Evaluate & Iterate

Use human + automated metrics to check alignment and stability

5️⃣ Self-Improvement Flywheel

Use current model π_t to generate and judge data → train π_{t+1}

Formal Objective (Bird's-Eye) 🧮

$$\max_{\theta}\ \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[R(x,y) - \beta\,\mathrm{KL}\big(\pi_\theta(\cdot|x)\,\|\,\pi_{\mathrm{ref}}(\cdot|x)\big)\right]$$

R(x, y): reward for response y on prompt x

β: KL penalty weight that controls how much the new policy can deviate from the reference

`π_ref`: the reference policy (usually the Supervised Fine-tuned model)

The KL term acts as a stability constraint - it prevents the model from drifting too far from the reference while still learning to maximize rewards

Where Does R(x, y) Come From? 🧪

Preference rewards: R_theta trained on pairwise (A > B)

AI-judge rewards: LLM Autorater

Verifiable rewards: unit tests (code), solvers (math), APIs (tools)

Process rewards: score intermediate reasoning steps

Evaluation at a Glance 🧪

Human eval: side-by-side preference, task-specific rubrics

Automated eval: `pass@k`

- Run the model's generated code k times with different samples
- `pass@1`: model must succeed on the 1st try
- `pass@k`: out of Top k sampled code completions, if ONE passes all test cases → count as success

Example

Task: Write a function is_even(n) that returns True if n is even.

LLM outputs (k = 3):

#1	return n % 2	❌
#2	return n % 2 == 0	✅
#3	return n % 2 != 1	✅

Results:

pass@1 → ❌ (#1 fails)

pass@2 → ✅ (at least one success among 2)

pass@3 → ✅ (at least one success among 3)

Recent Trends in RL for LLMs ⏩

Inexpensive scalable feedback via AI Autoraters

- Anthropic’s Constitutional AI show LLM-as-a-Judge can replace costly human raters
- Enables billions of preference pairs at low cost and consistent quality

Better automated verifiers for reasoning & tools

- Rise of verifiable rewards - external solvers, code testers, and symbolic checkers give objective pass/fail signals
- Used in math, code, and scientific reasoning tasks for stable RL updates

Recent Trends in RL for LLMs ⏩

o1 / Claude / Gemini Thinking models

- OpenAI o1 System Card - RL trains models “to reason using chain-of-thought”
- Anthropic’s Claude ‘Think’ Tool focus on process-level reasoning and verification

Convergence: RLHF → RLAIF → Verifiable RL → Self-Improving

- Together these advances mark scalable, verifiable, reasoning-driven RL

Data and Rewards

Three Data Regimes 🏗️

1️⃣ Human Preference (HF) - people compare A vs B

2️⃣ AI Feedback (AIF) - stronger model or rules judge outputs

3️⃣ Verifiable Signals - objective checkers return scores

Human Preference Data 🧍‍♀️

Prompt → (Response A, Response B) → Human label A ≻ B

Train a reward model `R_phi` to assign higher scores to preferred answers

Scale ≈ 10^5 – 10^6 pairs in modern datasets

Prompt: "Summarize the Reddit thread about commuting in NYC."

Response A: *3-sentence concise summary capturing key points*

Response B: *long, repetitive version*

Humans choose A ≻ B → `R_phi` learns human preference

Human Preference Data 🧍‍♀️

Pros:

- ✅ Captures genuine human values and intentions
- ✅ Produces high-quality, nuanced judgments (tone, empathy, clarity)

Cons:

- ⚠️ Subjective and labor-intensive
- ⚠️ Drift in annotator style over time
- ⚠️ Slow refresh → less suited for continual updates

AI Feedback & Constitutional Rules 🤖

Use LLMs as a rater/judge for responses

Common methods:

- RLAIF - GPT-5/Claude acts as evaluator for cheaper large-scale labeling
- Constitutional AI - judges responses using written “virtues”
- Self-Rewarding LMs - model critiques its own reasoning and generates preference data without human raters

AI Feedback & Constitutional Rules 🤖

Pros:

- ✅ Scalable - millions of comparisons at minimal cost
- ✅ Consistent - no fatigue or annotator drift
- ✅ Fast refresh - easy to regenerate data for new tasks

Cons:

- ⚠️ Style bias - judge prefers outputs written in its own phrasing
- ⚠️ Echo chamber - feedback loop reinforces same reasoning patterns

Verifiable Rewards ✅

Definition: reward can be checked by an external verifier

Examples:

- Math/Logic → symbolic solver
- Code → unit tests or compiler results
- QA → knowledge base lookup
- Tool use → API return values

Dense and objective → stable training and less bias

Verifiable Rewards ✅

"Anything that is verifiable can be directly optimized with RL."

Hybrid Data Pipelines 🔄

Mix multiple reward sources, and combine strengths of different feedback types:

- Human preference - captures values and empathy
- AI Feedback (scale) - provides large, consistent data cheaply
- Verifier (correctness) - ensures factual and logical accuracy

Curriculum approach

Start with broad AI feedback for coverage → End with strict verifiable checks for reliability and truth

Goal

Balance breadth in one training loop → diverse input from humans, AI judgment, or objective verification

Data Bias and Quality Issues ⚠️

Human annotator bias (verbosity, tone)

LLM judge bias (favors its own style)

Distribution drift between synthetic and real queries

Quality control practices

- Majority vote: aggregate several judges' opinions for stability
- Reward normalization: rescale scores to avoid runaway values
- Gold human subset: maintain a small, hand-checked dataset to monitor drift

Core Algorithms for RL in LLMs

Formal Objective (Bird's-Eye) 🧮

$$\max_{\theta} \; \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)} \left[R(x,y) - \beta\,\mathrm{KL}\big(\pi_\theta(\cdot|x) \,\|\, \pi_{\mathrm{ref}}(\cdot|x)\big)\right]$$

R(x, y): reward for response y on prompt x

β: KL penalty weight that controls how much the new policy can deviate from the reference

π_ref: the reference policy (usually the Supervised Fine-tuned model)

The KL term acts as a stability constraint - it prevents the model from drifting too far from the reference while still learning to maximize rewards

On-Policy vs Off-Policy 🔁

On-policy:

The same policy that generates data is the one being updated

Examples: PPO, A2C

Pros: stable, matches current distribution

Cons: needs new rollouts each update → expensive for LLMs

On-Policy vs Off-Policy 🔁

Off-policy:

Uses data from other policy models

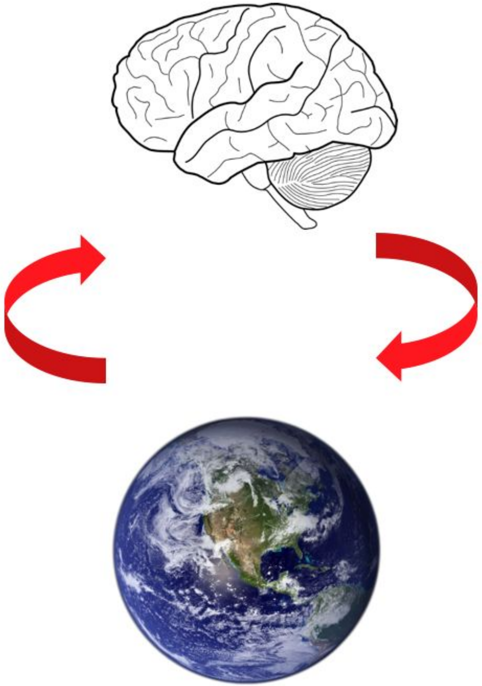
Examples: DPO, Q-learning

Pros: sample-efficient, can reuse old data

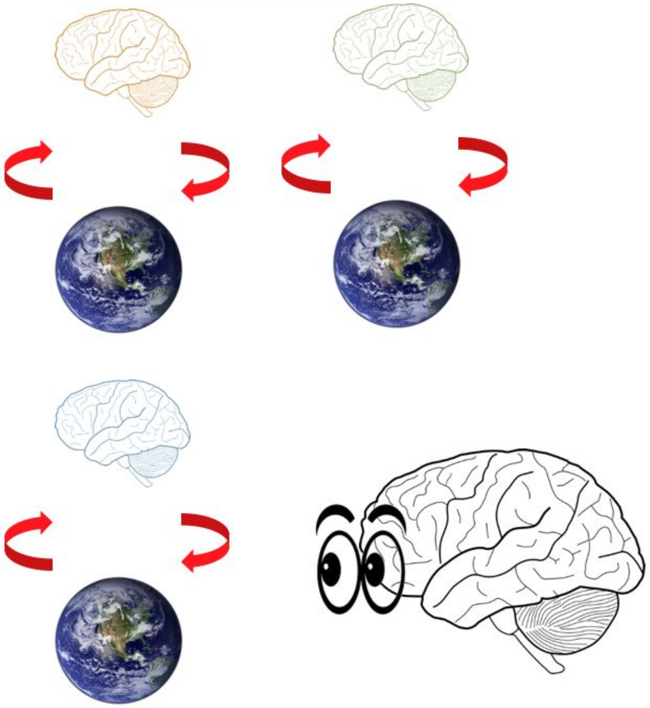
Cons: risk of mismatch between data and current policy

On-Policy vs Off-Policy 🔁

On-policy



Off-policy



On-Policy vs Off-Policy 🔁

Aspect	On-Policy	Off-Policy
Definition	Learns from data generated by the *current* policy being trained	Learns from data collected by *any* policy (past or other)
Data freshness	Always uses new samples each iteration	Reuses existing or replay-buffer data
Examples (LLMs)	PPO in RLHF	DPO, IPO
Advantages	Matches current behavior → low distribution mismatch; stable learning	Sample-efficient, cheaper, scalable; no need for new rollouts
Disadvantages	Costly - requires generating new responses; slower training	Possible data mismatch; relies on old or static preferences

PPO: Proximal Policy Optimization 🧩

Core idea: small, controlled policy updates

MAXIMIZE objective (simplified version):

$$L(\theta) = \mathbb{E}_t\Big[\mathrm{clip}\big(r_t(\theta), 1-\epsilon, 1+\epsilon\big)\, A_t\Big] \;-\; \beta\,\mathrm{KL}\big(\pi_\theta(a_t \mid s_t)\,\|\,\pi_{\mathrm{ref}}(a_t \mid s_t)\big)$$

$$\text{where} \quad r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\mathrm{old}}(a_t \mid s_t)}$$

Deep-dive: Advantage A_t 🧩

Intuition: Advantage `A_t` tells us “Is this action better OR worse than random action?”

- `A_t` > 0 → this action was better than expected
- `A_t` < 0 → this action was worse than expected

So PPO tries to:

- increase the probability of actions with positive advantage
- decrease the probability of actions with negative advantage

Toy Example 🪀

Prompt: “Explain what a binary tree is to a beginner.”

Action 1: clear, simple explanation → human/AI gives high score → $A_t \approx +2$

Action 2: overly formal, confusing answer → low score → $A_t \approx -1$

During training:

- For Action 1 → PPO pushes up its probability
- For Action 2 → PPO pushes down its probability

Over time, the model learns to prefer answers with consistently positive A_t

Deep-dive: Clip ✂️

In PPO we have a ratio:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{old}}(a_t \mid s_t)}$$

This says "how much did we change the probability of this action?"

- r_t > 1 → we increased its probability
- r_t < 1 → we decreased its probability

But we do **NOT** want huge jumps, even if A_t is large. That is where clipping comes in:

- We restrict r_t to a range like [1 - ε, 1 + ε], e.g. [0.8, 1.2]
- This implies: "You can change the probability, but not too much in one step."

Toy Example 🪀

Let ε = 0.2 → allowed range [0.8, 1.2]

Good action: $A_t = +2$

- Suppose $r_t = 1.8$ (big increase)
- Clipped to 1.2 → we still increase its probability, but not crazily

Bad action: $A_t = -1$

- Suppose $r_t = 0.4$ (huge decrease)
- Clipped to 0.8 → we still decrease its probability, but in a controlled way

Key idea for clipping:

Let us learn from advantage… but do NOT overreact in a single update

Deep-dive: KL divergence 🧩

$$\mathrm{KL}(\pi_\theta(\cdot \mid s_t) \parallel \pi_{\mathrm{ref}}(\cdot \mid s_t)) = \sum_a \pi_\theta(a \mid s_t) \log \frac{\pi_\theta(a \mid s_t)}{\pi_{\mathrm{ref}}(a \mid s_t)}$$

measures how different the new policy is from a reference policy (old model or SFT model)

- Small KL → behavior is similar to reference
- Large KL → behavior has drifted far away

In practice, we subtract β * KL from the objective:

If KL becomes too big, the loss punishes the model

This pushes the new policy back toward the reference

Putting It Together 🌈

Advantage `A_t`

- Says whether a particular answer was better or worse than usual
- Good A_t → probability up; bad A_t → probability down

Clipping

- Stops probability changes from being too extreme
- Keeps learning stable and prevents “one-batch disasters”

KL term

- Keeps the new model close to the reference model
- Preserves style, safety, and general behavior while optimizing reward

💡 PPO uses 1) advantage to know which actions to favor, 2) clipping to avoid over-updating, and 3) KL as a safety belt to keep the policy model from drifting too far while it learns

PPO in LLM Training 🧠

1. Sample prompts from dataset
2. Generate responses using current policy π
3. Get rewards from reward model
4. Update via PPO loss with rewards
5. Evaluate and repeat

DPO: Direct Preference Optimization 🔄

Observation: PPO needs expensive sampling and a separate reward model.

DPO simplifies by using preference pairs directly.

Loss:

$$L(\theta) = \mathbb{E}_{(x,\, y^+,\, y^-)} \left[\log \sigma\Big(\beta \left(\log \pi_\theta(y^+ \mid x) - \log \pi_\theta(y^- \mid x)\right)\Big)\right]$$

Intuition: increase probability of preferred outputs,

Pros:

- simpler, no reward model, no on-policy sampling

Cons:

- relies fully on quality of offline preference pairs

PPO vs DPO 🔁

Aspect	PPO (on-policy)	DPO (off-policy)
Data source	freshly generated	fixed dataset
Stability	high with KL control	very stable
Cost	high	low
Reward model needed?	yes	no
Typical use	RLxF	preference fine-tuning

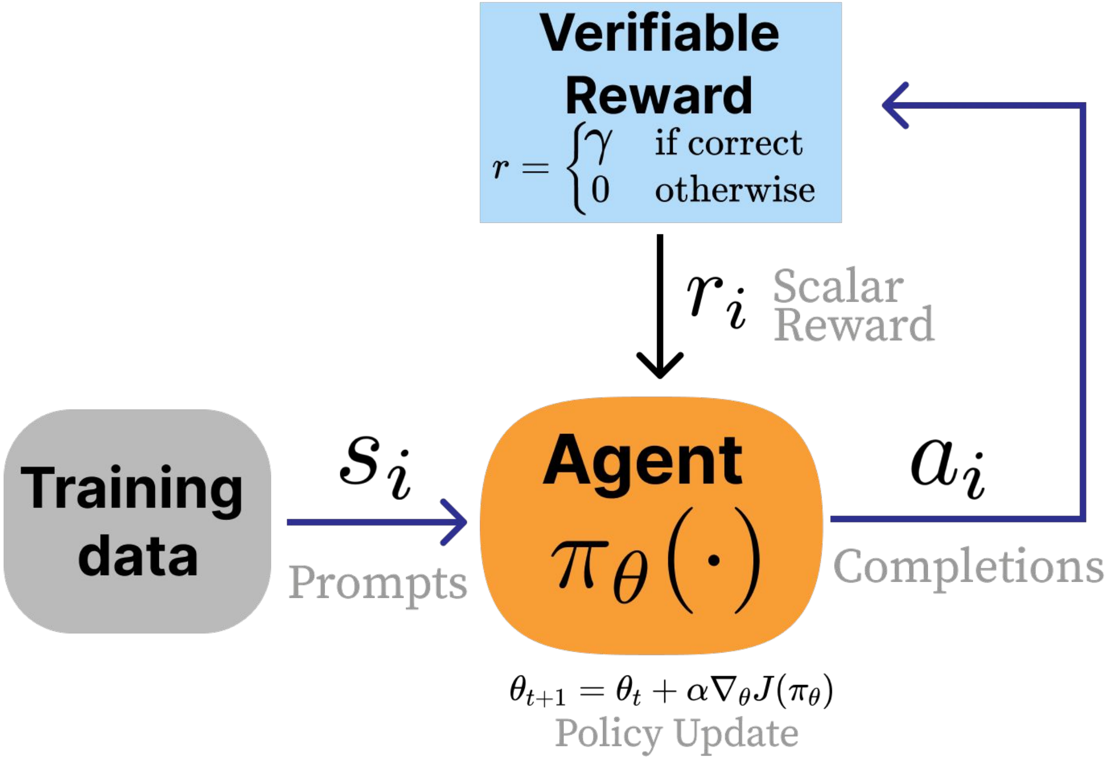
Beyond PPO/DPO: Recent Variants 🚀

Algorithm	Key Idea	Benefit
GRPO	Introduces gradient regularization to control update size	Improved stability and smoother convergence
IPO	Implicit Preference Optimization — reformulates preference loss as a unified probabilistic objective	Theoretical generalization of DPO/PPO family
RLAIF	Uses AI-judge feedback instead of human annotations	Highly scalable and consistent
Verifiable RL	Derives rewards from external checkers or tests (code, math, logic)	Improves reasoning accuracy and reliability

Verifiable RL Loop ✅

1. Sample prompt
2. Model generates output
3. External verifier checks correctness
4. Reward = verifier score (e.g., pass rate, logical validity)
5. Update policy via PPO or DPO objective

Verifiable RL Loop ✅



Putting It All Together 🔧

- PPO → on-policy, reward model, strong but costly
- DPO → off-policy, no reward model, simpler
- Verifiable RL → uses external correctness signals

All follow same principle:

maximize expected reward while staying close to reference behavior

RL for Thinking Improvements

From “Can Think” to “Think Better” 🚀

SFT (Stage 1)

Teaches models how to reason: generate chains of thought, follow examples

Output looks logical, but quality isn’t checked or rewarded

→ Model can think, but not always think well

From “Can Think” to “Think Better” 🚀

RL (Stage 2)

Adds evaluation and optimization on top of reasoning traces

Reward = external (verifier, human, AI judge) or self-generated score

Encourages concise, correct, verifiable reasoning paths

→ Model learns to reflect, self-correct, and improve its logic over time

Result

From passive imitation → active reasoning improvement

RL = feedback loop that refines thinking quality, not just output style

What is Self-Improvement? ⚙️

A policy π that generates its own training data and rewards

π□ produces outputs → scores them (via verifier or self-critique) → trains π□$_{+1}$

Each generation becomes a better teacher for the next

Data Generation Loop for Self-Improvement ♻️

1️⃣ Current policy π_t → generate responses

→ This creates fresh candidate data reflecting its current ability

2️⃣ Verifier or AI judge → score outputs

→ Converts outputs into quantitative rewards or preferences

3️⃣ Build new preference / verifiable dataset → train π_{t+1}

→ The policy learns from its own evaluated experience

4️⃣ Repeat for continuous refinement

→ Produces a self-evolving model that steadily improves reasoning and accuracy

Evolution of Learning Paradigms 🦕

Stage	Core Idea	What It Teaches the Model	Typical Methods	Key Outcome
Pretraining	Predict next token	Learn language, syntax, and world patterns	Self-supervised LM	🗣️ Fluent but not aligned
SFT	Learn from human instructions	Follow tasks, reason explicitly	Instruction tuning	💭 "Can Think"
RLHF / RLAIF	Learn from preferences	Align with human or AI values	PPO, DPO	🤝 "Thinks nicely"
Verifiable RL	Learn from objective signals	Seek truth and correctness	PPO / DPO + verifiers	✅ "Thinks correctly"
Self-Improvement	Learn from its own judgment	Reflect, verify, and evolve	RL flywheel	🔁 "Thinks better and grows"